aws

Guía para desarrolladores, versión 2

AWS IoT Greengrass

AWS IoT Greengrass: Guía para desarrolladores, versión 2

Table of Contents

¿Qué es AWS IoT Greengrass? 1
Nuevas características 1
Para nuevos usuarios 2
Para los usuarios ya existentes 2
Cómo AWS IoT Greengrass funciona 2
Conceptos clave 3
Características de AWS IoT Greengrass 5
Compatibilidad de características de Greengrass 7
Elegir el tiempo de ejecución de AWS IoT Greengrass Nucleus 16
Núcleo de Greengrass 17
Versión lite del núcleo de Greengrass 17
Novedades que incluye la versión 2 21
AWS IoT Greengrass Actualización de software Core v2.16.1 24
Actualizaciones de componentes públicos 24
AWS IoT Greengrass Actualización de software Core v2.16.0 25
Actualizaciones de componentes públicos 25
AWS IoT Greengrass Actualización de software Core v2.15.1 27
Actualizaciones de componentes públicos 28
AWS IoT Greengrass Actualización de software Core v2.15.0 29
Actualizaciones de componentes públicos 29
AWS IoT Greengrass Actualización de software Core v2.14.3 31
Actualizaciones de componentes públicos 31
AWS IoT Greengrass Actualización de software Core v2.14.2 33
Actualizaciones de componentes públicos 33
AWS IoT Greengrass Actualización de software Core v2.14.1 34
Actualizaciones de componentes públicos 35
AWS IoT Greengrass Actualización de software Core v2.14.0 36
Actualizaciones de componentes públicos 38
AWS IoT Greengrass Actualización de software Core v2.13.0 41
Actualizaciones de componentes públicos 41
AWS IoT Greengrass Actualización de software Core v2.12.6 43
Actualizaciones de componentes públicos 43
AWS IoT Greengrass Actualización de software Core v2.12.5 44
Actualizaciones de componentes públicos 45

AWS IoT Greengrass Actualización de software Core v2.12.4 46
Actualizaciones de componentes públicos 46
AWS IoT Greengrass Actualización de software Core v2.12.3 47
Actualizaciones de componentes públicos 47
AWS IoT Greengrass Actualización de software Core v2.12.2 49
Actualizaciones de componentes públicos 49
AWS IoT Greengrass Actualización de software Core v2.12.1 51
Actualizaciones de componentes públicos 51
AWS IoT Greengrass Actualización de software Core v2.12.0 53
Actualizaciones de componentes públicos 53
AWS IoT Greengrass Actualización de software Core v2.11.3 54
Actualizaciones de componentes públicos 54
AWS IoT Greengrass Actualización de software Core v2.11.2 56
Actualizaciones de componentes públicos 56
AWS IoT Greengrass Actualización de software Core v2.11.1 57
Actualizaciones de componentes públicos 57
AWS IoT Greengrass Actualización de software Core v2.11.0 59
Actualizaciones de componentes públicos 59
AWS IoT Greengrass Actualización de software Core v2.10.3 61
Actualizaciones de componentes públicos 61
AWS IoT Greengrass Actualización de software Core v2.10.2 62
Actualizaciones de componentes públicos 62
AWS IoT Greengrass Actualización de software Core v2.10.1 64
Actualizaciones de componentes públicos 64
AWS IoT Greengrass Actualización de software Core v2.10.0 66
Actualizaciones de componentes públicos 66
AWS IoT Greengrass Actualización de software Core v2.9.6 68
Actualizaciones de componentes públicos 68
AWS IoT Greengrass Actualización de software Core v2.9.5 69
Actualizaciones de componentes públicos 69
AWS IoT Greengrass Actualización de software Core v2.9.4 70
Actualizaciones de componentes públicos 71
AWS IoT Greengrass Actualización de software Core v2.9.3 72
Actualizaciones de componentes públicos 72
AWS IoT Greengrass Actualización de software Core v2.9.2 73
Actualizaciones de componentes públicos 73

AWS IoT Greengrass Actualización de software Core v2.9.1 74
Actualizaciones de componentes públicos 75
AWS IoT Greengrass Actualización de software Core v2.9.0 76
Actualizaciones de componentes públicos 77
AWS IoT Greengrass Actualización de software Core v2.8.1 79
Actualizaciones de componentes públicos 79
AWS IoT Greengrass Actualización de software Core v2.8.0 80
Actualizaciones de componentes públicos 81
AWS IoT Greengrass Actualización de software Core v2.7.0 83
Actualizaciones de componentes públicos 83
AWS IoT Greengrass Actualización de software Core v2.6.0 86
Actualizaciones de componentes públicos 87
AWS IoT Greengrass Actualización de software Core v2.5.6 91
Actualizaciones de componentes públicos 91
AWS IoT Greengrass Actualización de software Core v2.5.5 93
Actualizaciones de componentes públicos 93
AWS IoT Greengrass Actualización de software Core v2.5.4 94
Actualizaciones de componentes públicos 94
AWS IoT Greengrass Actualización de software Core v2.5.3 95
Actualizaciones de componentes públicos 96
AWS IoT Greengrass Actualización de software Core v2.5.2 97
Actualizaciones de componentes públicos 97
AWS IoT Greengrass Actualización de software Core v2.5.1 99
Actualizaciones de componentes públicos 99
AWS IoT Greengrass Actualización de software Core v2.5.0 100
Actualizaciones de compatibilidad con plataformas 101
Actualizaciones de componentes públicos 102
AWS IoT Greengrass Actualización de software Core v2.4.0 106
Actualizaciones de componentes públicos 107
AWS IoT Greengrass Actualización de software Core v2.3.0 109
Actualizaciones de componentes públicos 110
AWS IoT Greengrass Actualización de software Core v2.2.0 111
Actualizaciones de componentes públicos 112
AWS IoT Greengrass Actualización de software Core v2.1.0 115
Actualizaciones de compatibilidad con plataformas 116
Actualizaciones de componentes públicos 116

AWS IoT Greengrass Actualización de software Core v2.0.5 125
Actualizaciones de componentes públicos 125
AWS IoT Greengrass Actualización de software Core v2.0.4 126
Actualizaciones de componentes públicos 126
Migración desde la versión 1 128
Información general sobre la migración 128
Diferencias entre V1 y V2 130
Elija su tiempo de ejecución 142
Matriz de compatibilidad de fuentes de eventos 142
flujo de decisiones de selección en tiempo de ejecución 143
Siguientes pasos 144
Configurar el dispositivo de prueba 144
Configurar el dispositivo de prueba (núcleo Greengrass) 145
Configurar el dispositivo de prueba (Greengrass nucleus lite) 151
Actualización de los dispositivos principales de V1 a V2 220
Paso 1: Instale la versión AWS IoT Greengrass 2.x del software principal 221
Paso 2: Implementar componentes de Greengrass V2 en los dispositivos principales 224
Introducción 227
Requisitos previos 228
Paso 1: configurar una AWS cuenta 229
Inscríbase en una Cuenta de AWS 229
Creación de un usuario con acceso administrativo 230
Paso 2: Configurar el entorno 231
Paso 3: Instalar el software AWS IoT Greengrass principal 238
Instalación del software AWS IoT Greengrass Core (consola) 239
Instalación del software AWS IoT Greengrass Core (CLI) 245
Ejecución del software Greengrass (Linux) 250
Verificación de la instalación de la CLI de Greengrass en el dispositivo 251
Paso 4: Desarrollo y prueba de un componente en su dispositivo 253
Paso 5: Cree su componente en el AWS IoT Greengrass servicio 265
Paso 6: Implementación de su componente 277
Pasos a seguir a continuación 282
Configuración de los dispositivos principales de Greengrass 284
Plataformas admitidas 284
Requisitos de los dispositivos 285
Requisitos de la función de Lambda 285

Configura un Cuenta de AWS 287
Instalación del software AWS IoT Greengrass Core 288
Instalación con aprovisionamiento automático 292
Instalación con aprovisionamiento manual 308
Instalación con el aprovisionamiento de flotas 347
Instalación con aprovisionamiento personalizado 396
Argumentos del instalador 413
Ejecute el software AWS IoT Greengrass principal 418
Compruebe si el software AWS IoT Greengrass Core se ejecuta como un servicio del sistema 419
Ejecute el software AWS IoT Greengrass Core como un servicio del sistema 421
Ejecute el software AWS IoT Greengrass Core sin un servicio de sistema 421
Ejecute AWS IoT Greengrass en Docker 422
Plataformas compatibles y requisitos 423
Descargas de software 424
Elija cómo aprovisionar los recursos AWS 424
Cree la AWS IoT Greengrass imagen a partir de un Dockerfile 425
Se ejecuta AWS IoT Greengrass en Docker con aprovisionamiento automático 431
Se ejecuta AWS IoT Greengrass en Docker con aprovisionamiento manual 440
Solución de problemas AWS IoT Greengrass en un contenedor Docker 459
Configurar el software AWS IoT Greengrass principal 463
Implementación del componente núcleo de Greengrass 463
Configuración del núcleo de Greengrass como un servicio del sistema 463
Control de la asignación de memoria con las opciones de JVM 467
Configuración del usuario que ejecuta los componentes 470
Configuración de los límites de recursos del sistema 475
Realizar la conexión en el puerto 443 o a través de un proxy de red 477
Use un certificado de dispositivo firmado por una CA privada 486
Configuración de los tiempos de espera y los ajustes de caché de MQTT 486
Configurar Greengrass Nucleus en la red IPv6 486
Actualización del software AWS IoT Greengrass Core (OTA) 487
Requisitos 488
Consideraciones para los dispositivos principales 488
Comportamiento de actualización del núcleo de Greengrass 489
Cómo realizar actualizaciones OTA 491
Desinstalación del software AWS IoT Greengrass Core 491

Tutoriales 495
Tutoriales de vídeo 495
Videos 495
Desarrollo de un componente que aplace las actualizaciones de los componentes 496
Requisitos previos 497
Paso 1: Instalar la CLI del kit de desarrollo de Greengrass 499
Paso 2: Desarrollar un componente que aplace las actualizaciones 499
Paso 3: Publicar el componente en el AWS IoT Greengrass servicio 508
Paso 4: Implementación y prueba del componente en un dispositivo principal 512
Interacción con dispositivos IoT locales a través de MQTT 517
Requisitos previos 518
Paso 1: Revise y actualice la AWS IoT política principal de dispositivos 519
Paso 2: Habilitar la compatibilidad del dispositivo de cliente 521
Paso 3: Conectar los dispositivos de cliente 527
Paso 4: Desarrollar un componente que se comunique con los dispositivos de cliente 530
Paso 5: Desarrollar un componente que interactúe con las sombras de dispositivos de cliente 538
Proteja Greengrass Nucleus con TPM 565
Requisitos previos 565
Paso 1: Instalar TPM2 herramientas y dependencias 566
Paso 2: inicializar el almacén PKCS#11 y crear un slot 566
Paso 3: crear un token y una clave 568
Paso 4: genere una solicitud de firma de certificado (CSR) 570
Paso 5: crear el certificado de objetos 571
Paso 6: importar el certificado de objetos al TPM 571
Paso 7: capturar la URL del objeto PKCS#11 571
Paso 8: Configurar e instalar Greengrass con soporte TPM2 573
Paso 9: verificar las instalaciones 574
Resolución de problemas 575
Siguientes pasos 576
Proteja Greengrass Nucleus Lite con TPM 576
Requisitos previos 576
Paso 1: Configurar una instancia de NitroTPM 577
Paso 2: Instalar y configurar las herramientas de TPM 577
Paso 3: Configurar el proveedor de OpenSSL TPM2 578
Paso 4: generar claves TPM persistentes 578

Paso 5: generar una solicitud de firma de certificado (CSR) 579
Paso 6: Crea el certificado a partir de CSR 579
Paso 7: Configurar Greengrass Nucleus Lite con compatibilidad con TPM 580
Resolución de problemas 581
Siguientes pasos 582
Comience con SageMaker AI Edge Manager 582
Requisitos previos 583
Configúrelo en SageMaker AI Edge Manager 585
Creación de los componentes de muestra 587
Ejecución de un ejemplo de inferencia de clasificación de imágenes 588
Realización de una inferencia de clasificación de imágenes de muestra 592
Requisitos previos 593
Paso 1: Suscribirse al tema de notificaciones predeterminado 594
Paso 2: Implemente el componente de clasificación de imágenes TensorFlow Lite 595
Paso 3: Visualizar los resultados de la inferencia 596
Siguientes pasos 599
Realización de una inferencia de clasificación de imágenes de muestra en imágenes de una cámara 599
Requisitos previos 600
Paso 1: Configurar el módulo de cámara del dispositivo 601
Paso 2: Comprobar la suscripción al tema de notificaciones predeterminado 604
Paso 3: Modifique la configuración del componente de clasificación de imágenes de TensorFlow Lite e impleméntelo 604
Paso 4: Visualizar los resultados de la inferencia 606
Siguientes pasos 607
Componentes 608
Componentes proporcionados por AWS 608
Núcleo de Greengrass 628
Versión lite del núcleo de Greengrass 675
Autenticación del dispositivo de cliente 682
CloudWatch métricas 762
AWS IoT Device Defender 789
Spooler de disco 807
Administrador de aplicaciones de Docker 812
Conector periférico para Kinesis Video Streams 823
CLI de Greengrass 831

Detector de IP 847
Firehose 858
Lanzador de Lambda 878
Administrador de Lambda 882
Tiempos de ejecución de Lambda 892
Enrutador de suscripción antigua 895
Consola de depuración local 907
Administrador de registros 925
Componentes de machine learning 977
Adaptador de protocolo Modbus-RTU 1104
Puente MQTT 1138
Agente MQTT 3.1.1 (Moquette) 1163
Agente MQTT 5 (EMQX) 1171
Emisor de telemetría del núcleo 1189
Proveedor PKCS#11 1202
Administrador de secretos 1211
Tunelización segura 1225
Administrador de sombras 1237
Amazon SNS 1268
Administrador de flujos 1288
Reenviador de registros del sistema 1303
Agente de Systems Manager 1309
Servicio de intercambio de token 1317
Colector IoT SiteWise OPC UA 1321
Simulador de fuente de datos IoT SiteWise OPC UA 1331
SiteWise Publicador de IoT 1334
SiteWise Procesador IoT 1350
Componentes compatibles con Publisher 1366
AiShield.edge 1366
Sensor AI de EdgeLabs 1367
Greengrass S3 Ingestor 1368
Componentes de la comunidad 1368
Herramientas de desarrollo de Greengrass 1372
CLI del kit de desarrollo de Greengrass 1374
Interfaz de la línea de comandos de Greengrass 1406
Uso de Greengrass Testing Framework 1425

Desarrollo de componentes 1442
Componente del ciclo de vida 1444
Tipos de componentes 1445
Creación de componentes 1446
Prueba de los componentes con implementaciones locales 1459
Publicación de componentes a implementar 1462
Interacción con servicios de AWS 1468
Ejecución de un contenedor de Docker 1472
Referencia de receta 1496
Variables de entorno 1528
Implementación de componentes en los dispositivos 1530
Implementaciones en dispositivos principales 1530
Resolución de dependencias de plataformas 1530
Resolución de dependencias de componentes 1530
Eliminación de un dispositivo de un grupo de objetos 1538
Implementaciones 1539
Opciones de implementación 1540
Crear implementaciones 1542
Creación de subimplementaciones 1562
Revisión de las implementaciones 1566
Cancelación de implementaciones 1568
Comprobación del estado de la implementación 1569
Registro y monitoreo 1574
Herramientas de supervisión 1574
Supervisión de registros de Greengrass 1575
Acceso a los registros del sistema de archivos 1576
Acceso a los Registros de CloudWatch 1579
Acceso a los registros de servicios del sistema 1581
Habilitación del registro en los Registros de CloudWatch 1582
Configuración de registro en AWS IoT Greengrass 1584
AWS CloudTrailRegistros de 1586
Registra las llamadas a la API con CloudTrail 1586
AWS IoT Greengrass V2 información en CloudTrail 1587
AWS IoT Greengrass eventos de datos en CloudTrail 1588
AWS IoT Greengrass eventos de gestión en CloudTrail 1593
Descripción de las entradas de los archivos de AWS IoT Greengrass V2 registro 1593

Recopilación de datos de telemetría del estado del sistema 1594
Métricas de telemetría 1596
Configuración de los ajustes del agente de telemetría 1601
Suscríbase a los datos de telemetría en EventBridge 1602
Reciba notificaciones sobre el estado de la implementación y de los componentes 1610
Evento de cambio de estado de una implementación 1611
Evento de cambio de estado del componente 1613
Requisitos previos para crear reglas EventBridge 1615
Configuración de las notificaciones de estado del dispositivo (consola) 1616
Configuración de las notificaciones de estado del dispositivo (CLI) 1617
Configuración de las notificaciones de estado del dispositivo (CloudFormation) 1618
Véase también 1618
Comprobación del estado del dispositivo principal 1618
Comprobación del estado de un dispositivo principal 1619
Comprobación del estado de un grupo de dispositivos principales 1620
Comprobación del estado del componente del dispositivo principal 1620
Ejecución de las funciones de Lambda 1622
Requisitos 1623
Configuración del ciclo de vida de una función de Lambda 1623
Configuración de contenedores de funciones de Lambda 1625
Importación de una función de Lambda como componente (consola) 1627
Paso 1: Elegir una función de Lambda para importar 1628
Paso 2: Configurar los parámetros de la función de Lambda 1628
Paso 3: (Opcional) Especificar las plataformas compatibles con la función de Lambda 1631
Paso 4: (Opcional) Especificar las dependencias de los componentes para la función de Lambda 1632
Paso 5: (Opcional) Ejecutar la función de Lambda en un contenedor 1633
Paso 6: Crear el componente función de Lambda 1634
Importación de una función de Lambda como componente (CLI) 1635
Paso 1: Definir la configuración de la función de Lambda 1635
Paso 2: Crear el componente función de Lambda 1656
Comuníquese con el núcleo de Greengrass, otros componentes y AWS IoT Core 1658
Versiones de cliente de IPC 1659
Compatible SDKs 1660
Conéctese al AWS IoT Greengrass servicio Core IPC 1660
Autorización de los componentes para realizar operaciones de IPC 1666

Comodines en las políticas de autorización 1668
Variables de receta en las políticas de autorización 1668
Caracteres especiales en las políticas de autorización 1669
Ejemplos de políticas de autorización 1670
Suscripción a los flujos de eventos de IPC 1673
Definición de controladores de suscripción 1674
Ejemplos de controladores de suscripciones 1677
Prácticas recomendadas de IPC 1689
Publicar/suscribir mensajes locales 1690
Versiones mínimas de SDK 1691
Autorización 1691
PublishToTopic 1694
SubscribeToTopic 1704
Ejemplos 1721
Publicar/suscribir mensajes MQTT AWS IoT Core 1755
Versiones mínimas de SDK 1755
Autorización 1756
PublishToIoTCore 1761
SubscribeToIoTCore 1773
Ejemplos 1790
Interacción con el ciclo de vida del componente 1810
Versiones mínimas de SDK 1810
Autorización 1811
UpdateState 1812
SubscribeToComponentUpdates 1815
DeferComponentUpdate 1816
PauseComponent 1818
ResumeComponent 1819
Interacción con la configuración de componentes 1821
Versiones mínimas de SDK 1821
GetConfiguration 1822
UpdateConfiguration 1825
SubscribeToConfigurationUpdate 1828
SubscribeToValidateConfigurationUpdates 1833
SendConfigurationValidityReport 1834
Recupere valores secretos 1835

Versiones mínimas de SDK 1836
Autorización 1836
GetSecretValue 1838
Ejemplos 1844
Interactúe con las sombras locales 1850
Versiones mínimas de SDK 1851
Autorización 1851
GetThingShadow 1864
UpdateThingShadow 1871
DeleteThingShadow 1879
ListNamedShadowsForThing 1885
Administre las implementaciones y los componentes locales 1892
Versiones mínimas de SDK 1893
Autorización 1894
CreateLocalDeployment 1896
ListLocalDeployments 1900
GetLocalDeploymentStatus 1900
ListComponents 1901
GetComponentDetails 1902
RestartComponent 1904
StopComponent 1907
CreateDebugPassword 1907
Autenticación y autorización de los dispositivos de cliente 1908
Versiones mínimas de SDK 1909
Autorización 1910
VerifyClientDeviceIdentity 1912
GetClientDeviceAuthToken 1912
AuthorizeClientDeviceAction 1914
SubscribeToCertificateUpdates 1914
Interacción con dispositivos IoT locales 1917
Componentes de dispositivo de cliente 1918
Conexión de dispositivos de cliente a los dispositivos principales 1921
Requisitos 1922
Componentes de Greengrass para compatibilidad con dispositivos de cliente 1935
Configuración de la detección en la nube (consola) 1938
Configuración de la detección en la nube (AWS CLI) 1938

Asociación de los dispositivos de cliente 1938
Autenticación de clientes sin conexión 1941
Administración de puntos de conexión del dispositivo principal 1942
Elección de un agente MQTT 1950
Conexión a un agente de MQTT 1951
Probar las comunicaciones 1954
API de descubrimiento de Greengrass RESTful 1966
Retransmitir mensajes MQTT entre dispositivos cliente y AWS IoT Core 1973
Configuración e implementación del componente puente de MQTT 1973
Retransmisión de mensajes MQTT 1975
Interacción con los dispositivos de cliente en los componentes 1976
Configuración e implementación del componente puente de MQTT 1976
Recepción de mensajes MQTT desde los dispositivos de cliente 1978
Envío de mensajes MQTT a dispositivos de cliente 1978
Interacción y sincronización con las sombras de dispositivo de cliente 1979
Requisitos previos 1979
Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente 1980
Interacción con las sombras de dispositivo de cliente en los componentes 1983
Sincronice las sombras de los dispositivos cliente con AWS IoT Core 1983
IPv6 Úselo para mensajería local 1983
Configure el detector IP para usar IPv6 1984
Resolución de problemas 1988
Problemas de detección de Greengrass 1988
Problemas de conexión con MQTT 1997
Interacción con las sombras de dispositivo 2003
Interacción con las sombras de los componentes 2004
Recuperación y modificación de los estados de sombra 2004
Reacción a los cambios en el estado de sombra 2005
Sincronice las sombras de los dispositivos locales con AWS IoT Core 2006
Requisitos previos 2007
Configuración del componente administrador de sombras 2007
Sincronización de sombras locales 2009
Comportamiento conflictivo en la combinación de sombras 2010
Administre los flujos de datos 2011
Flujo de trabajo de la administración de secuencias 2012

Requisitos 2013
Seguridad de los datos 2014
Seguridad de los datos locales 2014
Autenticación del cliente 2014
Véase también 2015
Configurar el administrador de secuencias 2015
Parámetros del administrador de secuencias 2015
Véase también 2018
Creación de componentes personalizados que usen el administrador de flujos 2019
Definición de las recetas de componentes que utilizan el administrador de flujos 2019
Conexión al administrador de flujos en el código de la aplicación 2031
Se usa StreamManagerClient para trabajar con transmisiones 2034
Creación de una secuencia de mensajes 2036
Agregar un mensaje 2039
Lectura de mensajes 2046
Lista de secuencias 2048
Descripción de una secuencia de mensajes 2050
Actualización de un flujo de mensajes 2052
Eliminación de una secuencia de mensajes 2056
Véase también 2058
Exportación de configuraciones para destinos de nube compatibles 2058
Cómo realizar la inferencia de machine learning 2075
Cómo funciona la inferencia AWS IoT Greengrass de aprendizaje automático 2075
¿Qué hay de diferente en la AWS IoT Greengrass versión 2? 2077
Requisitos 2077
Orígenes de modelos admitidos 2078
Tiempos de ejecución admitidos 2078
Componentes de machine learning 2079
Utilice SageMaker AI Edge Manager 2087
Funcionamiento 2088
Requisitos 2089
Comience con SageMaker AI Edge Manager 2091
Personalización de sus componentes de machine learning 2091
Modificación de la configuración de un componente de inferencia público 2092
Uso de un modelo personalizado con el componente de inferencia de muestra 2094
Creación de componentes de machine learning personalizados 2098

Creación de un componente de inferencia personalizado 2101
Resolución de problemas 2108
No fue posible recuperar la biblioteca 2109
Cannot open shared object file 2110
Error: ModuleNotFoundError: No module named '<library>' 2110
No se ha detectado ningún dispositivo compatible con CUDA 2111
No existe tal archivo o directorio 2112
RuntimeError: module compiled against API version 0xf but this version of NumPy is <version> 2113
picamera.exc.PiCameraError: Camera is not enabled 2113
Errores de memoria 2114
Errores de espacio en el disco 2114
Errores de tiempo de espera 2114
Administración de los dispositivos principales con AWS Systems Manager 2115
Instalar el agente de Systems Manager 2116
Paso 1: completar los pasos generales de configuración de Systems Manager 2117
Paso 2: Crear un rol de servicio de IAM para Systems Manager 2117
Paso 3: Agregar permisos al rol de intercambio de token 2117
Paso 4: Implementar el componente agente de Systems Manager 2122
Paso 5: Verificar el registro del dispositivo principal en Systems Manager 2125
Desinstalación del agente de Systems Manager 2126
Paso 1: Anular el registro del dispositivo principal de Systems Manager 2127
Paso 2: Desinstalar el componente agente de Systems Manager 2127
Paso 3: Desinstalar el software agente de Systems Manager 2128
Seguridad 2129
Protección de datos 2130
Cifrado de datos 2131
Integración de la seguridad de hardware 2133
Autenticación y autorización de dispositivos 2146
Certificados X.509 2147
AWS IoT políticas 2148
Actualice la AWS IoT política de un dispositivo principal 2154
Política mínima de AWS IoT 2159
AWS IoT Política mínima de compatibilidad con los dispositivos cliente 2162
AWS IoT Política mínima para los dispositivos cliente 2164
Identity and Access Management 2166

Público 2166
Autenticación con identidades 2167
Administración del acceso con políticas 2168
Véase también 2170
Cómo AWS IoT Greengrass funciona con IAM 2170
Ejemplos de políticas basadas en identidades 2174
Autorizar a los dispositivos principales a interactuar con AWS los servicios 2177
Política de IAM mínima para que el instalador aprovisione recursos 2182
Rol de servicio de Greengrass 2186
AWS políticas gestionadas 2195
Prevención de la sustitución confusa entre servicios 2201
Solución de problemas de identidades y accesos 2202
Cómo permitir el tráfico del dispositivo a través de un proxy o firewall 2204
Puntos de conexión para el funcionamiento básico 2205
Puntos de conexión para la instalación con aprovisionamiento automático 2210
Puntos finales para los componentes AWS proporcionados 2212
Validación de conformidad 2212
Puntos de conexión de FIPS 2213
Habilitación de los puntos de conexión de FIPS con la implementación 2214
Instale el núcleo con puntos de conexión de FIPS con aprovisionamiento manual de recursos 2216
Instalación de puntos de conexión de FIPS con aprovisionamiento de flota 2250
Instalación de puntos de conexión de FIPS con aprovisionamiento automático de recursos 2269
Componentes propios que cumplen con las normas FIPS 2286
Resiliencia 2287
Seguridad de la infraestructura 2288
Configuración y análisis de vulnerabilidades 2288
Integridad del código 2289
Puntos de conexión de VPC (AWS PrivateLink) 2291
Consideraciones sobre los puntos AWS IoT Greengrass finales de VPC 2291
Cree un punto final de VPC de interfaz para las operaciones del plano AWS IoT Greengrass de control 2292
Crear una política de puntos de conexión de VPC para AWS IoT Greengrass 2292
Opere un dispositivo AWS IoT Greengrass central en VPC 2293
Prácticas recomendadas de seguridad 2298

Conceda los mínimos permisos posibles 2299
No codifique las credenciales en los componentes de Greengrass 2299
No registre información confidencial 2299
Mantenga sincronizado el reloj del dispositivo 2300
Recomendaciones de conjunto de cifrado 2300
Véase también 2300
Uso de AWS IoT Device Tester para la versión 2 de AWS IoT Greengrass 2301
Guía de cualificación de AWS IoT Greengrass 2301
Compatibilidad con los conjuntos de prueba 2302
Versiones compatibles 2302
Última versión de IDT para V2 AWS IoT Greengrass 2303
Versiones anteriores de IDT para AWS IoT Greengrass 2303
Versiones no AWS IoT Device Tester compatibles AWS IoT Greengrass de para V2 2304
Descarga de IDT para AWS IoT Greengrass V2 2310
Descarga de IDT manualmente 2310
Descarga de IDT mediante programación 2311
Utilice IDT para ejecutar el conjunto de AWS IoT Greengrass cualificaciones 2317
Versiones del conjunto de pruebas 2317
Descripciones de los grupos de pruebas 2318
Requisitos previos 2321
Configuración de su dispositivo para ejecutar pruebas de IDT 2343
Configuración de los ajustes de IDT 2354
Ejecute el conjunto AWS IoT Greengrass de calificaciones 2372
Descripción de los resultados y de los registros 2375
Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas 2379
Descargar la última versión de IDT para AWS IoT Greengrass 2321
Flujo de trabajo de creación de conjuntos de prueba 2380
Tutorial: crear y ejecutar el ejemplo del conjunto de pruebas de IDT 2381
Tutorial: Desarrollar un conjunto de pruebas de IDT sencillo 2386
Creación de archivos de configuración para el conjunto de pruebas de IDT 2395
Configuración del orquestador de pruebas de IDT 2404
Configure la máquina de estados IDT 2411
Cree ejecutables de casos de prueba de IDT 2436
Uso del contexto de IDT 2444
Configuración de los ajustes para los ejecutores de pruebas 2449
Depurar y ejecutar conjuntos de pruebas personalizadas 2461

Revisión de los resultados y registros de las pruebas de IDT 2464
Métricas de uso de IDT 2470
Solución de problemas de IDT para V2 AWS IoT Greengrass 2478
¿Dónde puedo buscar los errores? 2478
Resolución de errores de IDT para la V2 AWS IoT Greengrass 2479
Política de soporte de AWS IoT Device Tester para AWS IoT Greengrass 2487
Soluciones de IoT basadas en Greengrass 2488
Eurotech 2488
Resolución de problemas 2489
Consulte los registros AWS IoT Greengrass principales de software y componentes 2489
AWS IoT Greengrass Problemas principales de software 2489
ThrottlingException desde ListDeployments la API 2491
No se ha podido configurar el dispositivo principal 2491
No se puede iniciar el software AWS IoT Greengrass Core como un servicio del sistema .. 2491
No se puede configurar el núcleo como un servicio del sistema 2491
No se puede conectar a AWS IoT Core 2492
Error de falta de memoria 2492
No se puede instalar la CLI de Greengrass 2493
User root is not allowed to execute 2493
com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with 2493
Failed to map segment from shared object: operation not permitted 2494
No se pudo configurar el servicio de Windows 2494
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager 2495
com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime 2495
software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid 2495
software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy 2496
Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request 2497
Operation aws.greengrass#<operation> is not supported by Greengrass 2498
java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied) 2498

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist 2498
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn> . 2499
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed 2500
java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi 2500
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED 2500
Greengrass core device stuck on nucleus v2.12.3 2501
Greengrass nucleus v2.14.0 systemd template issue 2503
AWS IoT Greengrass problemas con la nube 2504
An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null 2504
Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed} 2504
INACTIVE deployment status 2505
Problemas de implementación del dispositivo principal 2505
Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact 2506
Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption. 2507
Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements> 2508
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility 2509
com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component 2509

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service 2510
Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration 2511
Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy 2512
Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration 2512
Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null) 2513
Problemas con los componentes del dispositivo principal 2513
Warn: '<command>' is not recognized as an internal or external command 2514
El script de Python no registra los mensajes 2514
La configuración de los componentes no se actualiza al cambiar la configuración predeterminada 2515
awsiot.greengrasscoreipc.model.UnauthorizedError 2517
com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>" 2517
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400) 2518
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403) 2520
com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers 2520
Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>" 2521
copyFrom: <configurationPath> is already a container, not a leaf 2521
com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.' 2522
java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired. 2522
aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant 2524

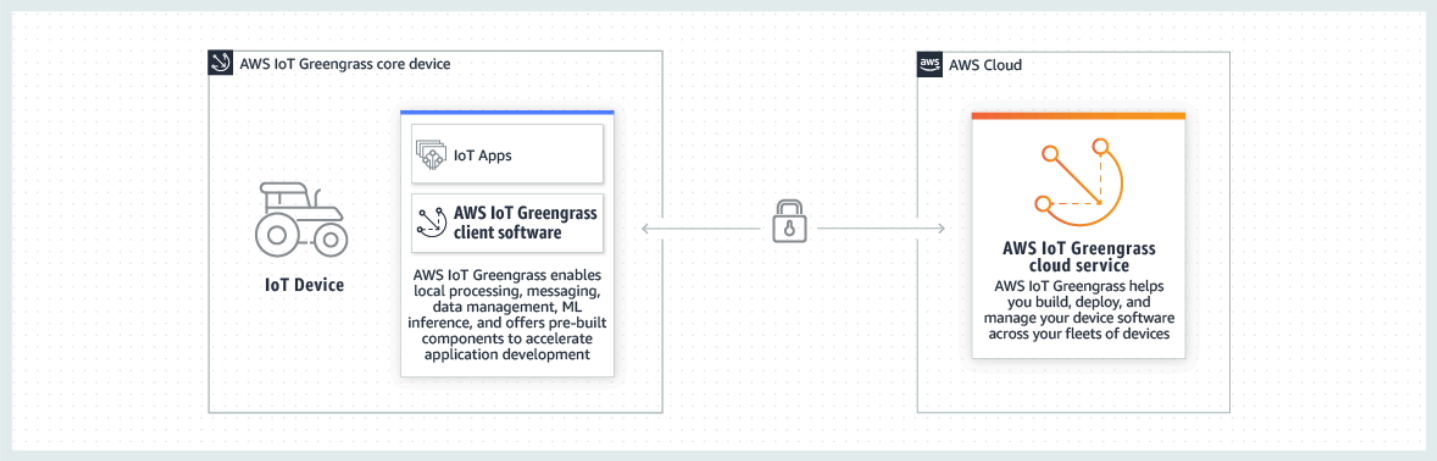
Problemas con los componentes de la función de Lambda del dispositivo principal 2524
The following cgroup subsystems are not mounted: devices, memory 2524
ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn> 2525
Versión del componente descontinuada 2525
Problemas con la CLI de Greengrass 2526
java.lang.RuntimeException: Unable to create ipc client 2526
AWS CLI problemas 2527
Error: Invalid choice: 'greengrassv2' 2527
Códigos de error de implementación detallados 2527
Error de permiso 2529
Error de solicitud 2530
Error en la receta del componente 2533
AWS error de componente, error de componente de usuario, error de componente 2535
Error del dispositivo 2536
Error de dependencia 2538
Error de HTTP 2539
Error de red 2539
Error de núcleo 2540
Error del servidor 2541
Error del servicio en la nube 2542
Errores genéricos 2543
Error desconocido 2544
Códigos de estado de componentes detallados 2544
Etiquetar los recursos 2548
Uso de etiquetas en AWS IoT Greengrass V2 2548
Etiquete con Consola de administración de AWS 2548
Etiquete con la AWS IoT Greengrass V2 API 2548
Uso de etiquetas con políticas de IAM 2550
AWS CloudFormation recursos 2552
AWS IoT Greengrass y CloudFormation plantillas 2552
ComponentVersion ejemplo de plantilla 2552
Ejemplo de plantilla de implementación 2553
Obtenga más información sobre CloudFormation 2554
Software de código abierto 2555
Historial de revisión 2556

AWS Glosario 2630
..... mmdcxxxi

¿Qué es AWS IoT Greengrass?

AWS IoT Greengrass es un servicio en la nube y de tiempo de ejecución perimetral del Internet de las cosas (IoT) de código abierto que le ayuda a crear, implementar y administrar aplicaciones de IoT en sus dispositivos. Puede utilizarlo AWS IoT Greengrass para crear software que permita a sus dispositivos actuar de forma local a partir de los datos que generan, ejecutar predicciones basadas en modelos de aprendizaje automático y filtrar y agregar los datos de los dispositivos. AWS IoT Greengrass permite que sus dispositivos recopilen y analicen datos más cerca de donde se generan, reaccionen de forma autónoma ante los eventos locales y se comuniquen de forma segura con otros dispositivos de la red local. Los dispositivos Greengrass también pueden comunicarse de forma segura con los datos de IoT AWS IoT Core y exportarlos a. Nube de AWS Puede utilizarlos AWS IoT Greengrass para crear aplicaciones perimetrales mediante módulos de software prediseñados, denominados componentes, que pueden conectar sus dispositivos perimetrales a AWS servicios o servicios de terceros. También puede utilizarlo AWS IoT Greengrass para empaquetar y ejecutar el software mediante funciones Lambda, contenedores de Docker, procesos nativos del sistema operativo o tiempos de ejecución personalizados de su elección.

El siguiente ejemplo muestra cómo interactúa un AWS IoT Greengrass dispositivo con. Nube de AWS



Nuevas características

AWS IoT Greengrass V2 presenta nuevas funciones y mejoras. A continuación, se incluye más información sobre las nuevas características que se ofrecen en la versión 2.

- Qué hay de nuevo en AWS IoT Greengrass Version 2

Para usuarios primerizos de AWS IoT Greengrass

Si es la primera vez que lo usa AWS IoT Greengrass, le recomendamos que consulte la siguiente sección:

- Cómo AWS IoT Greengrass funciona

A continuación, sigue el tutorial de introducción para probar las funciones básicas de AWS IoT Greengrass. En este tutorial, instalará el software AWS IoT Greengrass principal en un dispositivo, desarrollará un componente de Hello World y empaquetará ese componente para su implementación.

Para los usuarios actuales de AWS IoT Greengrass V1

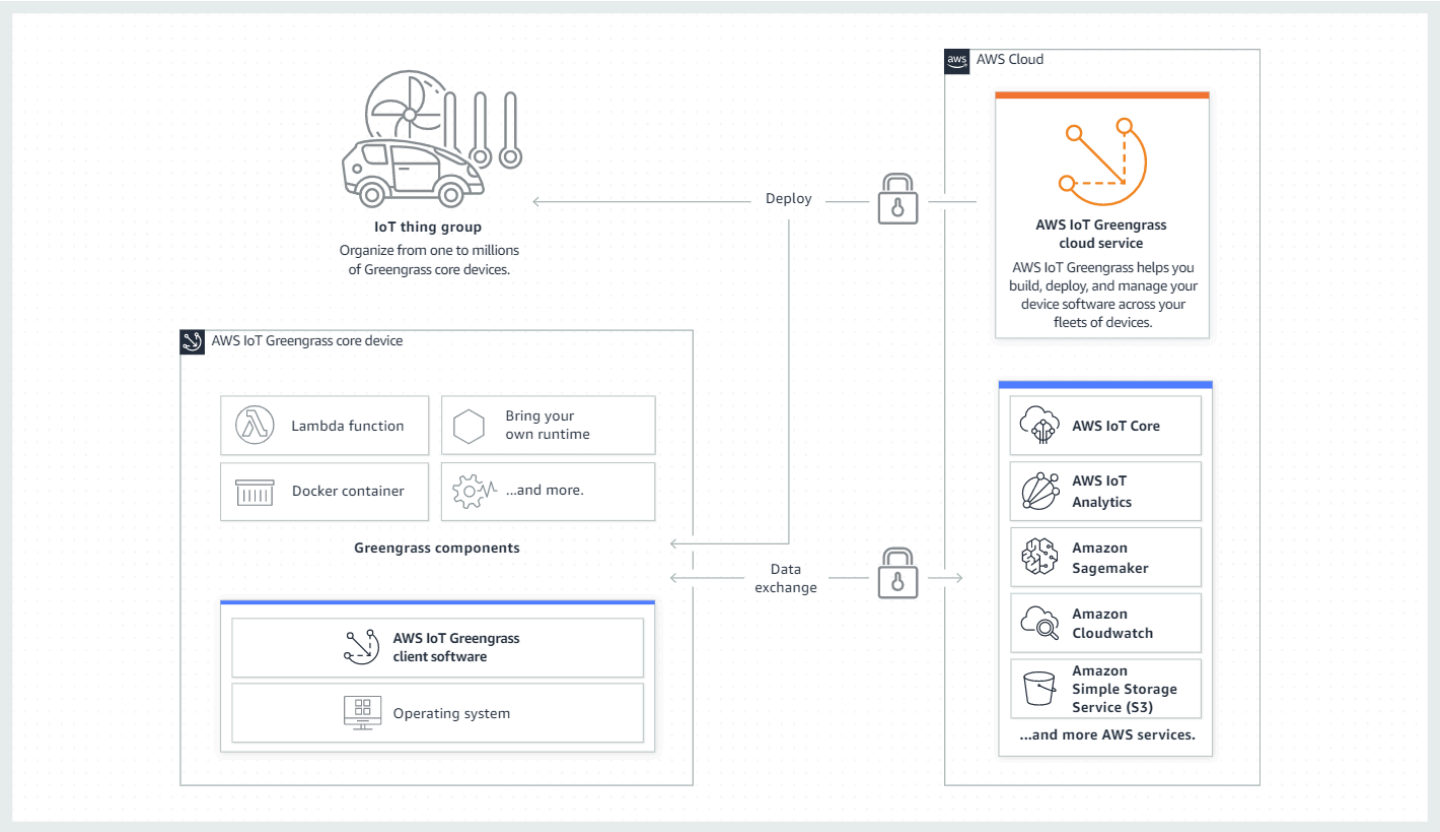
Aviso de fin del soporte: el 7 de octubre de 2026 AWS finalizará el soporte para AWS IoT Greengrass Version 1. Después del 7 de octubre de 2026, ya no podrás acceder a la AWS IoT Greengrass V1 consola ni a AWS IoT Greengrass V1 los recursos. Para obtener más información sobre cómo pasar de la versión 1 a la versión 2, consulte Migrar desde AWS IoT Greengrass la versión 1.

Cómo AWS IoT Greengrass funciona

El software AWS IoT Greengrass cliente, también denominado software AWS IoT Greengrass Core, se ejecuta en distribuciones basadas en Windows y Linux, como Ubuntu o Raspberry Pi OS, para dispositivos con arquitecturas ARM o x86. Con él AWS IoT Greengrass, puede programar los dispositivos para que actúen localmente a partir de los datos que generan, ejecutar predicciones basadas en modelos de aprendizaje automático y filtrar y agregar los datos de los dispositivos. AWS IoT Greengrass permite la ejecución local de AWS Lambda funciones, contenedores Docker, procesos nativos del sistema operativo o tiempos de ejecución personalizados de su elección.

AWS IoT Greengrass proporciona módulos de software prediseñados denominados componentes que le permiten ampliar fácilmente la funcionalidad de los dispositivos periféricos. AWS IoT Greengrass los componentes le permiten conectarse a AWS servicios y aplicaciones de terceros en la periferia. Después de desarrollar sus aplicaciones de IoT, AWS IoT Greengrass le permite implementar, configurar y administrar esas aplicaciones de forma remota en su flota de dispositivos sobre el terreno.

El siguiente ejemplo muestra cómo interactúa un AWS IoT Greengrass dispositivo con el servicio en la AWS IoT Greengrass nube y otros AWS servicios del Nube de AWS.



Conceptos clave para AWS IoT Greengrass

Los siguientes son conceptos esenciales para la comprensión y el uso de AWS IoT Greengrass:

AWS IoT cosa

Una AWS IoT cosa es una representación de un dispositivo específico o entidad lógica. La información sobre una cosa se almacena en el AWS IoT registro.

DIspositivo principal de Greengrass

Dispositivo que ejecuta el software AWS IoT Greengrass Core. Un dispositivo central de Greengrass es cosa del AWS IoT. Puede añadir varios dispositivos principales a grupos de AWS IoT cosas para crear y gestionar grupos de dispositivos principales de Greengrass. Para obtener más información, consulte Configuración de los dispositivos AWS IoT Greengrass principales.

Dispositivo de cliente de Greengrass

Un dispositivo que se conecta y se comunica con un dispositivo principal de Greengrass a través de MQTT. Un dispositivo cliente de Greengrass existe. AWS IoT El dispositivo principal puede procesar, filtrar y agregar datos de los dispositivos de cliente que se conectan a él. Puede

configurar el dispositivo principal para retransmitir mensajes MQTT entre los dispositivos cliente, el servicio AWS IoT Core en la nube y los componentes de Greengrass. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Los dispositivos de cliente pueden ejecutar FreeRTOS o usar el SDK para dispositivos con AWS IoT o la API de detección de Greengrass para obtener información sobre los dispositivos principales a los que se pueden conectar.

Componente de Greengrass

Un módulo de software que se implementa y se ejecuta en un dispositivo principal de Greengrass. Todo el software con el que se desarrolla e implementa AWS IoT Greengrass se modela como un componente. AWS IoT Greengrass proporciona componentes públicos prediseñados que proporcionan características y funcionalidades que puede utilizar en sus aplicaciones. También puede desarrollar sus propios componentes personalizados, en su dispositivo local o en la nube. Después de desarrollar un componente personalizado, puede usar el servicio en la nube de AWS IoT Greengrass para implementarlo en uno o varios dispositivos principales. Puede crear un componente personalizado e implementar ese componente para un dispositivo principal. Al hacerlo, el dispositivo principal descarga los siguientes recursos para ejecutar el componente:

- Receta: un archivo JSON o YAML que describe el módulo de software definiendo los detalles, la configuración y los parámetros de los componentes.
- Artefacto: el código de origen, los binarios o los scripts que definen el software que se ejecutará en el dispositivo. Puede crear artefactos desde cero o puede crear un componente mediante una función de Lambda, un contenedor de Docker o un tiempo de ejecución personalizado.
- Dependencia: la relación entre los componentes que permite aplicar actualizaciones o reinicios automáticos de los componentes dependientes. Por ejemplo, puede hacer que un componente de procesamiento seguro de mensajes dependa de un componente de cifrado. Esto garantiza que cualquier actualización del componente de cifrado actualice y reinicie automáticamente el componente de procesamiento de mensajes.

Para obtener más información, consulte Componentes proporcionados por AWS y Desarrollo de componentes de AWS IoT Greengrass.

Implementación

El proceso para enviar componentes y aplicar la configuración de componentes deseada a un dispositivo objetivo de destino, que puede ser un único dispositivo principal de Greengrass o un grupo de dispositivos principales de Greengrass. Las implementaciones aplican automáticamente cualquier configuración de componentes actualizada al destino e incluyen cualquier otro

componente que se defina como dependencias. También puede clonar una implementación existente para crear una nueva que utilice los mismos componentes, pero que se implemente en un destino diferente. Las implementaciones son continuas, lo que significa que cualquier actualización que realice en los componentes o en la configuración de los componentes de una implementación se envía automáticamente a todos los destinos. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

AWS IoT Greengrass Software básico

A partir de la versión 2.14, AWS IoT Greengrass ofrece dos implementaciones alternativas del tiempo de ejecución de su dispositivo, un ejecutable conocido como núcleo. El primer núcleo, y anteriormente el único, está implementado en Java. Esta opción proporciona la mayor portabilidad entre arquitecturas y sistemas operativos. Sin embargo, también depende de la máquina virtual de Java, lo que supone un gran consumo de memoria.

El segundo núcleo, recientemente agregado, está implementado en C. Esta elección reduce su espacio en gran cantidad. Sin embargo, requiere una distribución (o compilación desde el código origen) por separado para las diferentes arquitecturas de objetivo y sistemas operativos. Cuando sea necesario distinguir las dos, nos referiremos a la primera implementación como núcleo de Greengrass y a la segunda como núcleo lite de Greengrass.

- Componentes opcionales: estos componentes configurables los proporcionan los dispositivos AWS IoT Greengrass perimetrales, que habilitan funciones adicionales. En función de sus requisitos, puede elegir los componentes opcionales que desee implementar en su dispositivo, como la transmisión de datos, la inferencia de machine learning local o una interfaz de línea de comandos local. Para obtener más información, consulte Componentes proporcionados por AWS.

Puede actualizar su software AWS IoT Greengrass principal implementando nuevas versiones de sus componentes en su dispositivo.

Características de AWS IoT Greengrass

AWS IoT Greengrass Version 2 consta de los siguientes elementos:

- Distribuciones de software
  - El componente núcleo de Greengrass, que es la instalación mínima del software AWS IoT Greengrass Core. Este componente administra las implementaciones, la orquestación y la administración del ciclo de vida de los componentes de Greengrass.

  - Componentes proporcionados por AWS adicionales y opcionales que se integran con los servicios, protocolos y software.
  - Herramientas de desarrollo de Greengrass, que puede utilizar para crear, probar, compilar, publicar e implementar componentes personalizados de Greengrass.
  - The SDK para dispositivos con AWS IoT, que contiene la biblioteca de comunicación entre procesos (IPC) para componentes personalizados de Greengrass y la biblioteca de descubrimiento de Greengrass para dispositivos cliente.
  - El SDK del administrador de flujos, que puede utilizar para administrar los flujos de datos en los dispositivos principales.
- Servicio en la nube
  - AWS IoT Greengrass V2 API
  - AWS IoT Greengrass V2 consola

AWS IoT Greengrass Software básico

Puede usar el software AWS IoT Greengrass Core que se ejecuta en sus dispositivos perimetrales para hacer lo siguiente:

- Procesar los flujos de datos en los dispositivos locales con exportaciones automáticas a la nube de AWS . Para obtener más información, consulte Administración de flujos de datos en los dispositivos principales de Greengrass.
- Support MQTT mensajería entre componentes AWS IoT y. Para obtener más información, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.
- Interactuar con los dispositivos locales que se conectan y se comunican a través de MQTT. Para obtener más información, consulte Interacción con dispositivos IoT locales.
- Admitir mensajes de publicación y suscripción locales entre componentes. Para obtener más información, consulte Publicar/suscribir mensajes locales.
- Implementar e invocar componentes y funciones de Lambda. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.
- Administrar los ciclos de vida de los componentes, por ejemplo, con soporte para scripts de instalación y ejecución. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.
- Realice actualizaciones de software seguras over-the-air (OTA) del software AWS IoT Greengrass principal y de los componentes personalizados. Para obtener más información, consulte

Actualización del software AWS IoT Greengrass Core (OTA) y Implemente AWS IoT Greengrass componentes en los dispositivos.

- Brindar almacenamiento seguro y cifrado de secretos locales y acceso controlado por componentes. Para obtener más información, consulte Administrador de secretos.
- Proteja las conexiones entre los dispositivos y la AWS nube con la autenticación y autorización de los dispositivos. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

Puede configurar y administrar los dispositivos principales de Greengrass mediante los AWS IoT Greengrass APIs cuales crea despliegues continuos de software. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Algunas características solo son compatibles con determinadas plataformas. Para obtener más información, consulte Compatibilidad de características de Greengrass.

Para obtener más información sobre plataformas compatibles, requisitos y descargas, consulte Configuración de los dispositivos AWS IoT Greengrass principales.

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

Compatibilidad de características de Greengrass

AWS IoT Greengrass admite dispositivos que ejecutan varios sistemas operativos. Algunas características solo son compatibles con algunos sistemas operativos. Utilice las siguientes tablas para saber qué características están disponibles para cada sistema operativo compatible. Para obtener más información acerca de los sistemas operativos compatibles, los requisitos y cómo configurar los dispositivos principales de Greengrass, consulte Configuración de los dispositivos AWS IoT Greengrass principales.

Mensajería

Característica	Linux	Windows	Greengrass Nucleus Lite (Linux)
Intercambie mensajes MQTT entre componentes y AWS IoT	 Sí	 Sí	 Sí

Característica	Linux	Windows	Greengrass Nucleus Lite (Linux)
			(excepto las MQTT5 extensiones)
Intercambia publish/subscribe mensajes locales entre componentes	Sí	Sí	Sí
Interacción con dispositivos IoT locales a través de MQTT	Sí	Sí	No
Interacción con dispositivos Modbus-RTU locales mediante el componente Modbus-RTU	Sí	No	No

Seguridad

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Conexiones seguras con autenticación y autorización de dispositivos	Sí	Sí	Sí
Implemente secretos cifrados y seguros y acceda a ellos desde AWS Secrets Manager	Sí	Sí	No

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Utilice un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado del dispositivo	 Sí	 No	Sí
Audite los dispositivos principales con AWS IoT Device Defender	 Sí	 Sí	 No
Utilice AWS las credenciales para interactuar con AWS los servicios	 Sí	 Sí	 Sí

Instalación

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Instale AWS IoT Greengrass con aprovisionamiento automático	 Sí	 Sí	 No
Instale AWS IoT Greengrass con aprovisionamiento manual	 Sí	 Sí	 Sí

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Instale AWS IoT Greengrass con el aprovisionamiento AWS IoT de flotas	 Sí	 Sí	 Sí
Instale AWS IoT Greengrass con aprovisionamiento personalizado	Sí	Sí	 Sí
Se ejecuta AWS IoT Greengrass en un contenedor de Docker con una imagen de Docker prediseñada	 Sí	 No	 No

> **Note**
>
> AWS IoT Greengrass se puede instalar y ejecutar en un contenedor docker habilitado para systemd.

Mantenimiento y actualizaciones remotos

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Realice actualizaciones de software seguras over-the-air (OTA)	 Sí	 Sí	 Sí

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Administre los dispositivos principales con AWS Systems Manager	 Sí	 No	 No
Conéctese a los dispositivos principales con una tunelización AWS IoT segura	 Sí	 No	 Sí

Machine learning

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Realice inferencias de aprendizaje automático con Amazon SageMaker AI Edge Manager	 Sí	 Sí	 No
Realización de inferencias de machine learning con Amazon Lookout for Vision	 Sí	 No	 No
Realice inferencias de machine learning utilizando DLR	 Sí	 Sí	 No

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Realice una inferencia de aprendizaje automático mediante TensorFlow	 Sí	 Sí	 No

Características del componente

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Implementación e invocación de funciones de Lambda	 Sí	 No	 No
Ejecute contenedores de Docker en los componentes	 Sí	 Sí	 Sí
Procese y exporte flujos de datos de gran volumen mediante el administrador de flujos	 Sí	 Sí	 Sí
Administre los ciclos de vida de los componentes con scripts de ciclo de vida	 Sí	 Sí	 Sí

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Interacción con las sombras de dispositivo	 Sí	 Sí	 No
Subir registros a Amazon CloudWatch Logs	 Sí	 Sí	 Sí
Sube datos a CloudWatch las métricas de Amazon mediante el componente de CloudWatch métricas	 Sí	 Sí	 Sí
Publicación de mensajes de evento en un tema de Amazon Simple Notification Service (Amazon SNS)	 Sí	 No	 No
Publique datos en las transmisiones de entrega de Amazon Kinesis Data Streams mediante Stream Manager	 Sí	 Sí	 Sí

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Publique datos en flujos de entrega de Amazon Data Firehose mediante el componente de Firehose	 Sí	 No	 No
Recopile las métricas de telemetría del sistema en tiempo real y actúe en función de ellas	 Sí	 Sí	 No
Configure los límites de recursos del sistema para los componentes	 Sí	 No	 No
Pause y reanude los procesos de los componentes	 Sí	 No	 No
Intégrelo con el AWS IoT SiteWise uso de los componentes AWS IoT SiteWise	 Sí	 Sí	 No

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Publique flujos de video en Amazon Kinesis Video Streams mediante el conector de periferia para el component e Kinesis Video Streams	 Sí	 No	 No

Desarrollo de componentes

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
Desarrolle component es localmente en los dispositivos principal es	 Sí	 Sí	 Sí
Interactúe con un dispositivo principal mediante la AWS IoT Greengrass CLI	 Sí	 Sí	 No
Interactúe con un dispositivo principal mediante la consola de depuración local	 Sí	 Sí	 No
Utilice SDK para dispositivos con AWS IoT para Python	 Sí	 Sí	 Sí

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
en componentes personalizados			
Utilice el SDK para dispositivos con AWS IoT para C++ en componentes personalizados	 Sí	 Sí	 Sí
Utilice el SDK para dispositivos con AWS IoT para Java en componentes personalizados	 Sí	 Sí	 Sí

Certificaciones de dispositivos

Característica	Linux	Windows	Versión lite de Greengrass (Linux)
AWS IoT Device Tester Úselo AWS IoT Greengrass V2 para validar dispositivos de IoT	 Sí	 Sí	 No

Elegir el tiempo de ejecución de AWS IoT Greengrass Nucleus

A partir de la versión 2.14.0, AWS IoT Greengrass ofrece dos implementaciones alternativas del tiempo de ejecución de su dispositivo, un ejecutable conocido como núcleo. A pesar de sus diferencias de implementación, ambos tiempos de ejecución son compatibles con el AWS IoT Greengrass servicio APIs y permiten implementar componentes proporcionados por el SDK de Greengrass AWS o desarrollar componentes personalizados mediante el SDK de Greengrass.

Además, según sea necesario, es posible mezclar dispositivos con cualquier tipo de núcleo dentro de la misma flota.

Sin embargo, para lograr la portabilidad deseada o los beneficios específicos de ahorro de memoria, es fundamental asegurarse de que el núcleo implementado en sus dispositivos de Greengrass sea compatible con los componentes que piensa utilizar con el fin de acelerar el desarrollo de las soluciones de AWS IoT . Para obtener más información sobre la compatibilidad de los componentes, consulte Components.

La elección entre las dos opciones del tiempo de ejecución de Greengrass dependerá de su caso de uso específico, las restricciones del dispositivo, los requisitos de características y el sistema operativo.

Núcleo de Greengrass

AWS IoT Greengrass nucleus es un entorno de ejecución con todas las funciones que le permite ejecutar AWS IoT Greengrass en una amplia gama de dispositivos, incluidos portales, servidores y dispositivos periféricos con más recursos de cómputo. Considere la posibilidad de elegir el núcleo de Greengrass si:

- Recursos informáticos: su dispositivo tiene suficientes recursos informáticos, como más de 128 MB de RAM y un procesador relativamente potente (por ejemplo, más de 1 GHz reloj).
- Se necesita compatibilidad total con el sistema operativo: Greengrass nucleus es compatible con la más amplia gama de sistemas operativos (incluidas la mayoría de las distribuciones de Linux y Windows).
- Compatibilidad de componentes: Greengrass nucleus ofrece la máxima compatibilidad con los componentes existentes publicados por el equipo de AWS IoT servicio y los socios.

Versión lite del núcleo de Greengrass

AWS IoT Greengrass nucleus lite es un motor de ejecución ligero y de código abierto que le permite funcionar AWS IoT Greengrass en dispositivos con recursos limitados. Esto puede ayudar a las computadoras que tienen una placa única de bajo costo con aplicaciones de gran volumen, como centros domésticos inteligentes, medidores de energía inteligentes, vehículos inteligentes, IA avanzada y robótica. Evalúe usar la versión lite del núcleo de Greengrass si sus dispositivos tienen:

- Recursos limitados: el dispositivo tiene recursos limitados, como memoria RAM (512 MB o menos), espacio de almacenamiento (FLASH) o un procesador de bajo rendimiento (menos de 1). GHz

- Dependencia limitada: la plataforma de software del proveedor de su dispositivo no es compatible con Java ni con la JVM específica requerida por Greengrass Nucleus.
- Sistema operativo: sus dispositivos ejecutan una distribución de Linux compatible con systemd (por ejemplo, Ubuntu o Yocto).

Limitaciones actuales de la versión lite del núcleo de Greengrass

Como se incluye en la AWS IoT Greengrass versión 2.14.0, el tiempo de ejecución de Greengrass nucleus lite (v.2.0.0) ofrece un subconjunto de las funciones disponibles en Greengrass nucleus (v2.14.0).

El mecanismo AWS IoT Greengrass IPC (comunicación entre procesos) permite que los componentes se comuniquen con el núcleo de Greengrass. La versión ligera del núcleo es compatible con el siguiente subconjunto:

Característica	Disponibilidad.
SubscribeToTopic	Disponible
PublishToTopic	Disponible
PublishToIoTCore	Disponible
SubscribeToIoTCore	Disponible
UpdateState	No está disponible
SubscribeToComponentUpdates	No está disponible
DeferComponentUpdate	No está disponible
GetConfiguration	Disponible
UpdateConfiguration	Disponible
SubscribeToConfigurationUpdate	Disponible
SubscribeToValidateConfigurationUpdates	No está disponible.
SendConfigurationValidityReport	No está disponible.

Característica	Disponibilidad.
GetSecretValue	No está disponible.
PutComponentMetric	No está disponible
GetComponentDetails	No está disponible
RestartComponent	No está disponible
StopComponent	No está disponible
CreateLocalDeployment	Disponible
CancelLocalDeployment	No está disponible
GetLocalDeploymentStatus	No está disponible
ListLocalDeployments	No está disponible
ListComponents	No está disponible
ValidateAuthorizationToken	Disponible
CreateDebugPassword	No está disponible
PauseComponent	No está disponible
ResumeComponent	No está disponible
GetThingShadow	No está disponible
UpdateThingShadow	No está disponible
DeleteThingShadow	No está disponible
ListNamedShadowsForThing	No está disponible
SubscribeToCertificateUpdates	No está disponible
VerifyClientDeviceIdentity	No está disponible

Característica	Disponibilidad.
GetClientDeviceAuthToken	No está disponible
AuthorizeClientDeviceAction	No está disponible

Qué hay de nuevo en AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 es una versión principal AWS IoT Greengrass que presenta las siguientes funciones:

- AWS IoT Greengrass nucleus lite: AWS IoT Greengrass las versiones 2.14.0 y posteriores de AWS IoT Greengrass nucleus ofrecen nucleus lite. La versión lite del núcleo de Greengrass es un tiempo de ejecución ligero y de código abierto que permite ejecutar Greengrass en dispositivos con recursos limitados. Esto puede ayudar a las computadoras que tienen una placa única de bajo costo con aplicaciones de gran volumen, como centros domésticos inteligentes, medidores de energía inteligentes, vehículos inteligentes, IA avanzada y robótica. Para obtener más información, consulte Versión lite del núcleo de Greengrass.
- Componentes compatibles con Publisher: AWS IoT Greengrass ahora ofrece componentes compatibles con Publisher. Proveedores externos desarrollan, ofrecen y mantienen estos componentes. Para obtener más información, consulte Componentes compatibles con Publisher.
- Operar un dispositivo de Greengrass en VPC: ahora está disponible el funcionamiento de un dispositivo principal de Greengrass en VPC. Esto le permite realizar implementaciones en una VPC sin acceso público a Internet. Para obtener más información, consulte Opere un dispositivo AWS IoT Greengrass central en VPC.
- Greengrass Testing Framework(GTF): ahora está disponible GTF para AWS IoT Greengrass Version 2 . El GTF es un conjunto de componentes básicos que respaldan la automatización. end-to-end Permite a los clientes AWS IoT Greengrass Version 2 internos utilizar el mismo marco de pruebas que utiliza el equipo de servicio para calificar los cambios de software, aceptarlos automáticamente y garantizar la calidad. Para obtener más información, consulte Greengrass Testing Framework en Github.
- Certificación PSA: las versiones 2.7.0 y posteriores de AWS IoT Greengrass Nucleus cuentan ahora con la certificación Platform Security Architecture (PSA). Para obtener más información, consulte AWS IoT Greengrass cuenta con la certificación de PSA.

AWS IoT Greengrass las notas de la versión proporcionan detalles sobre las AWS IoT Greengrass versiones: nuevas funciones, actualizaciones y mejoras, y correcciones generales. AWS IoT Greengrass tiene los siguientes tipos de versiones:

- Lanzamientos de nuevas funciones para AWS IoT Greengrass
- AWS IoT Greengrass Actualizaciones de software principales

Esta sección contiene todas las notas de la AWS IoT Greengrass V2 versión, primero las más recientes, e incluye los principales cambios en las funciones y correcciones de errores importantes. Para obtener información sobre otras correcciones menores, consulte la organización de aws-greengrass en. GitHub

Notas de la versión

- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.16.1 el 23 de diciembre de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.16.0 el 6 de noviembre de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.15.1 el 6 de noviembre de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.15.0 el 3 de julio de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.3 el 11 de abril de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.2 el 27 de marzo de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.1 el 7 de febrero de 2025
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.0 el 16 de diciembre de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.13.0 el 26 de agosto de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.6 el 24 de mayo de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.5 el 25 de abril de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.4 el 2 de abril de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.3 el 27 de marzo de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.2 el 15 de febrero de 2024
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.1 el 8 de diciembre de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.0 el 7 de noviembre de 2023

- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.3 el 18 de octubre de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.2 el 9 de agosto de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.1 el 21 de julio de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.0 el 28 de junio de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.3 el 21 de junio de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.2 el 5 de junio de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.1 el 11 de mayo de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.0 el 9 de mayo de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.6 el 20 de abril de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.5 el 30 de marzo de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.4 el 24 de febrero de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.3 el 1 de febrero de 2023
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.2 el 22 de diciembre de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.1 el 18 de noviembre de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.0 el 15 de noviembre de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.8.1 el 13 de octubre de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.8.0 el 7 de octubre de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.7.0 el 28 de julio de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.6.0 el 27 de junio de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.6 el 31 de mayo de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.5 el 6 de abril de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.4 el 23 de marzo de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.3 el 6 de enero de 2022
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.2 el 3 de diciembre de 2021

- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.1 el 23 de noviembre de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.0 el 12 de noviembre de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.4.0 el 3 de agosto de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.3.0 el 29 de junio de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.2.0 el 18 de junio de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.1.0 el 26 de abril de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.0.5 el 9 de marzo de 2021
- Lanzamiento: actualización del software AWS IoT Greengrass Core v2.0.4 el 4 de febrero de 2021

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.16.1 el 23 de diciembre de 2025

Esta versión incluye la versión 2.16.1 del componente núcleo de Greengrass y la versión 2.16.1 del componente CLI de Greengrass.

Fecha de lanzamiento: 23 de diciembre de 2025

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de

actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.16.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Añade una configuración para los intervalos de reintento de credencia les tras un error en el servicio de intercambio de fichas.
CLI de Greengrass	Está disponible la versión 2.16.1 de la CLI de Greengrass. Versión actualizada para la versión 2.16.1 de Greengrass Nucleus.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.16.0 el 6 de noviembre de 2025

Esta versión incluye la versión 2.16.0 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados. AWS

Fecha de lanzamiento: 6 de noviembre de 2025

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que

las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.16.0 del núcleo de Greengrass. Nuevas características • Nucleus ahora es compatible con cgroups V2. Mejoras y correcciones de errores • Soluciona un problema por el que Nucleus no limpiaba las implement aciones obsoletas cuando se cancelaba una implementación.
Versión lite del núcleo de Greengrass	Está disponible la versión 2.3.0 del núcleo lite de Greengrass. Nuevas características • Support para usar TPM 2.0 para la autorización MQTT de IoT Core. • Los paquetes apt de muestra ahora son compatibles con más sistemas operativos: Ubuntu 22.04, Ubuntu 24.04, Debian 12 y Debian 13. • RestartComponent Ahora se admite el IPC. Mejoras y correcciones de errores • Las implementaciones locales ya no necesitan acceso a Internet. • GetConfiguration se ha actualizado para que coincida con el comportam iento en tiempo de ejecución de Greengrass Nucleus. (Un cambio radical) • Corrección de errores y mejoras generales.

Componente	Detalles
Administrador de sombras	Está disponible la versión 2.3.12 del administrador de sombras. Mejoras y correcciones de errores • Soluciona un problema por el que las implementaciones se bloqueaban cuando se configuraban más de 1024 dispositivos ocultos.
Administrador de registros	Está disponible la versión 2.3.11 del administrador de registros. Mejoras y correcciones de errores • Soluciona un problema por el que la configuración del tiempo de ejecución de Log Manager aumentaba indefinidamente debido a que la información de los archivos de registro cargados estaba obsoleta.
Reenviador de registros del sistema	Está disponible la versión 2.1.0 del reenviador de registros del sistema. Mejoras y correcciones de errores • Actualiza la receta del componente para que sea compatible correctam ente con el núcleo de Greengrass. • Se ha mejorado el resultado del registro cuando no hay registros que cargar. • Corrección de errores y mejoras generales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.15.1 el 6 de noviembre de 2025

Esta versión incluye la versión 2.15.1 del componente núcleo de Greengrass y la versión 2.15.1 del componente CLI de Greengrass.

Fecha de lanzamiento: 6 de noviembre de 2025

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass Versión 2.15.1	Mejoras y correcciones de errores • Soluciona un problema por el que Nucleus no podía limpiar los procesos de los componentes después del tiempo de espera de 30 segundos durante las implementaciones de arranque.
CLI de Greengrass Versión 2.15.1	Versión actualizada para la versión 2.15.1 de Greengrass Nucleus.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.15.0 el 3 de julio de 2025

Esta versión incluye la versión 2.15.0 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados. AWS

Fecha de lanzamiento: 3 de julio de 2025

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.15.0 del núcleo de Greengrass.

Componente	Detalles
	Nuevas características • Agrega una característica de telemetría para incluir información del sistema anfitrión, como detalles de la CPU y el sistema operativo. Para obtener más información, consulte métricas de telemetría. • Agrega la configuración `deploymentConfigurationTimeSource` . Para obtener más información, consulte la Configuración del núcleo de Greengrass. Mejoras y correcciones de errores • El núcleo ahora prioriza, por defecto, el uso de versiones y artefactos locales, lo que mejora la coherencia de la implementación y reduce las dependencias externas. Los usuarios pueden anular este comportam iento especificando preferencias alternativas en el documento de implementación. • Agrega un nuevo registro de advertencias cuando un usuario de Windows proporcionado no coincide con el conjunto de caracteres que normalmente acepta Windows.
CLI de Greengrass	Está disponible la versión 2.15.0 de la CLI de Greengrass. Mejoras y correcciones de errores • Agrega una solución para que, cuando la configuración de autenticación cambie, el componente no se vuelva a instalar, sino que se reinicie. • Corrección de errores y mejoras generales.
Versión lite del núcleo de Greengrass	Ya está disponible la versión 2.2.0 de la versión lite del núcleo de Greengrass. Nuevas características • Añade compatibilidad con el artefacto de imagen del contenedor. URIs Mejoras y correcciones de errores • Corrección de errores y mejoras generales.

Componente	Detalles
Administrador de registros	Ya está disponible la versión 2.3.10 del administrador de registros. Nuevas características • Agrega una nueva clave de configuración (`updateToTlogIntervalSec`) para controlar la frecuencia con la que los detalles de los eventos de carga de registros se conservan en el registro de transacciones local (`config.tlog`). Mejoras y correcciones de errores • Mejora el administrador de registros para actualizar el cliente de CloudWatch en caso de errores en la conexión del socket.
Administrador de secretos	Está disponible la versión 2.2.6 del administrador secreto. Mejoras y correcciones de errores • Soluciona un problema que impedía que el administrador de secretos obtuviera un secreto debido a que un módulo de plataforma confiable era lento o no respondía.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.3 el 11 de abril de 2025

Esta versión incluye la versión 2.14.3 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados. AWS

Fecha de lanzamiento: 11 de abril de 2025

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

Important

Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.14.3 del núcleo de Greengrass. Mejoras y correcciones de errores • Permite que el servicio de intercambio de tokens se reinicie al producirs e cambios en la configuración del puerto. • Soluciona un problema que impedía que el servicio de estado de la flota enviara mensajes de cambio de estado de los componentes para Lambdas no fijadas y que no se activaban. • Soluciona un problema que impedía que los componentes se apagaran correctamente cuando se implementaba una nueva versión del componente. • Soluciona un problema que provocaba que los enlaces de inyección de los complementos integrados se ejecutaran dos veces, lo que hacía que se produjeran eventos adicionales durante el ciclo de vida y se duplicara el registro. • Mejora el registro del ciclo de vida de los componentes en los dispositi vos Windows.

Componente	Detalles
CLI de Greengrass	Está disponible la versión 2.14.3 de la CLI de Greengrass. Mejoras y correcciones de errores • Versión actualizada para la versión 2.14.3 de Greengrass Nucleus.
Administrador de secretos	Está disponible la versión 2.2.4 del administrador secreto. Mejoras y correcciones de errores • Reduce la frecuencia de las escrituras en el almacén de secretos local. El administrador de secretos ahora escribe en la tienda local solo cuando se actualizan los secretos.
Agente MQTT de EMAX	Ya está disponible la versión 2.0.3 del agente MQTT de EMQX. Mejoras y correcciones de errores • Soluciona un problema que impedía a EMQX iniciar en Windows si la ruta contenía espacios.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.2 el 27 de marzo de 2025

Esta versión incluye la versión 2.14.2 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados. AWS

Fecha de lanzamiento: 27 de marzo de 2025

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.14.2 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema que impedía que un cliente HTTP estuviera configurado con una autenticación mutua.
CLI de Greengrass	Está disponible la versión 2.14.2 de la CLI de Greengrass. Mejoras y correcciones de errores • Versión actualizada para la versión 2.14.2 de Greengrass Nucleus.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.1 el 7 de febrero de 2025

Esta versión incluye la versión 2.14.1 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados. AWS

Fecha de lanzamiento: 7 de febrero de 2025

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.14.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema que provocaba que los componentes no se detuvieran correctamente en las nuevas instalaciones de Greengrass.
CLI de Greengrass	Está disponible la versión 2.14.1 de la CLI de Greengrass. Mejoras y correcciones de errores • Versión actualizada para la versión 2.14.1 de Greengrass Nucleus.

Componente	Detalles
Tunelización segura	Ya está disponible la versión 1.1.1 del componente de tunelización segura. Mejoras y correcciones de errores • Agrega una configuración compatible con la versión lite del núcleo de Greengrass.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.14.0 el 16 de diciembre de 2024

Esta versión proporciona la versión 2.14.0 del componente del núcleo de Greengrass y nuevas actualizaciones de la versión lite del núcleo de AWS IoT Greengrass . El AWS IoT Greengrass núcleo lite es un nuevo motor de ejecución, disponible para la AWS IoT Greengrass versión 2. Es una alternativa que ocupa menos memoria. Una buena opción para dispositivos con recursos limitados. Implementa un subconjunto de las funciones del núcleo con una mayor compatibilidad prevista para futuras versiones. Ya está disponible el código origen en GitHub. Con el tiempo de ejecución de la versión lite del núcleo de Greengrass, se puede realizar lo siguiente:

- Implementar componentes en los dispositivos principales de Greengrass. Utilizar el mismo formato de receta, aunque algunas características avanzadas todavía pueden que no estén disponibles.
- Las aplicaciones implementadas como componentes de Greengrass pueden usar el dispositivo SDKs para acceder al IPC de Greengrass compatible APIs, como el acceso AWS IoT Core MQTT, el pub/sub local y el acceso a la configuración de Greengrass. Consulte la tabla de compatibilidad para ver la lista de IPC compatibles. APIs
- Algunos componentes AWS gestionados se han actualizado para que sean compatibles con nucleus lite. Consulte los componentes proporcionados por AWS para obtener una lista de los componentes compatibles.

Nuevas características:

- Utiliza menos memoria y espacio en disco (menos de 5 MB de RAM y menos de 5 MB de almacenamiento).
- Los componentes se integran con el administrador de servicios del sistema host (systemd para las plataformas Linux compatibles).

Aspectos a tener en cuenta:

- AWS IoT Greengrass Las recetas de nucleus lite distinguen mayúsculas de minúsculas. Verifique que la carcasa (llaves) correcta se utiliza como en la referencia de la receta https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html.
- El tiempo de ejecución de la versión lite del núcleo admite implementaciones del grupo de objetos, pero aún no admite el tipo de destino de implementación (único) del dispositivo principal. Para realizar la implementación en un único dispositivo de Greengrass, utilice un grupo de objetos que contenga solo ese dispositivo.
- El tiempo de ejecución de la versión lite del núcleo utiliza recursos de memoria limitados. La funcionalidad se adapta según el uso en la versión clásica del tiempo de ejecución, la cual puede fallar debido a que se superan los recursos disponibles en la versión lite. Esto incluye una limitación actual de un máximo de 50 suscripciones a MQTT a la vez y límites máximos de tamaño e implementación de los archivos de recetas. Algunos de estos límites se pueden configurar en tiempo de compilación si usted compila el tiempo de ejecución de la versión lite.
- El tiempo de ejecución de la versión lite del núcleo no se incluye con Java. Para usar componentes que requieran Java, el sistema necesitará tener Java ya instalado, o bien se podrá utilizar un componente para instalar Java.
- Recomendamos compilar el tiempo de ejecución de la versión lite del núcleo desde la fuente y utilizar una compilación propia adaptada a su sistema. En el caso de los sistemas Yocto, ya está disponible una capa para integrar el tiempo de ejecución de la versión lite del núcleo en la imagen del sistema.
- Por ahora, la versión lite del núcleo asume que un sistema Linux utiliza systemd o que una imagen de contenedor utiliza systemd.
- Si bien puede administrar los contenedores de Docker con scripts de recetas, aún no están disponibles los artefactos de contenedores administrados por Greengrass.
- El motor de ejecución de nucleus lite aún no admite las claves almacenadas en un PKCS11 módulo. Si su caso de uso requiere que las claves se almacenen en un elemento seguro, la versión clásica del tiempo de ejecución clásico podrá admitirlas. Para evitar que se pierdan las credenciales de su dispositivo, asegúrese de que los dispositivos de producción utilicen un cifrado de disco completo.

Además, lanzaremos la versión 2.14.0 del núcleo junto con la introducción de la versión lite del núcleo. Esta actualización aporta mejoras significativas al núcleo actual de Greengrass.

Características y mejoras claves:

- El nuevo soporte para terminales de doble pila permite la comunicación en IPv6 red.
- Resiliencia mejorada contra los errores de reinicio del núcleo y la corrupción de directorios.
- Se corrigieron las fugas de memoria en los cierres de PubSub suscripciones a IPC.

Fecha de lanzamiento: 16 de diciembre de 2024

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Versión lite del núcleo de Greengrass	Ya está disponible la versión 2.0.0 de la versión lite del núcleo de Greengrass. Nuevas características • Utiliza menos memoria y espacio en disco (menos de 5 MB de RAM y menos de 5 MB de almacenamiento).

Componente	Detalles
	• Los componentes se integran con el administrador de servicios del sistema anfitrión (systemd para las plataformas Linux compatibles).
Núcleo de Greengrass	Está disponible la versión 2.14.0 del núcleo de Greengrass. Nuevas características • El nuevo soporte para terminales de doble pila permite la comunicación en red. IPv6 • Resiliencia mejorada contra los errores de reinicio del núcleo y la corrupción de directorios . Mejoras y correcciones de errores • Se corrigieron las fugas de memoria en los cierres de PubSub suscripci ones a IPC. • Soluciona el ciclo de vida de ejecución del componente, donde entra en estado de ERROR debido al tiempo de espera de inicio cuando la condición skipif es verdadera. • Soluciona un problema por el que el dispositivo principal no se puede conectar AWS IoT Core cuando la política de TLS está configurada en _1_3_2022_10. TLS13
CLI de Greengrass	Está disponible la versión 2.14.0 de la CLI de Greengrass. Mejoras y correcciones de errores • Valide el parámetro de destino de implementación en el comando CLI.
Administrador de flujos	Ya está disponible la versión 2.14.0 de administrador de flujos. Nuevas características • Agrega una nueva clave de configuración para el tiempo de espera de inicio. El valor predeterminado es 120 segundos. • Agregar compatibilidad de recetas para la versión lite del núcleo de Greengrass

Componente	Detalles
Agente MQTT 5 (EMQX)	Ya está disponible la versión 2.0.2 del agente MQTT 5 de (EMQX). Mejoras y correcciones de errores • Soluciona un problema que iniciaba EMQX antes de que el componente de autenticación del dispositivo cliente estuviera listo.
Componente de los tiempos de ejecución de Lambda	Ya está disponible la versión 2.0.9 del componente de los tiempos de ejecución de Lambda. Mejoras y correcciones de errores • Soluciona una advertencia de sintaxis con Python 3.12
Componente administrador de Lambda	Está disponible la versión 2.3.5 del componente Lambda Manager. Mejoras y correcciones de errores • Mejora el rendimiento al utilizar epoll en lugar de nio cuando está disponible
Componente administrador de secretos	Está disponible la versión 2.2.2 del componente Secret Manager. Mejoras y correcciones de errores • Soluciona un problema que impedía que el administrador de secretos descargara los secretos configurados con ARN parciales.
Componente de tunelización segura	Ya está disponible la versión 1.1.0 del componente de tunelización segura. Nuevas características • Agregar compatibilidad de recetas para la versión lite del núcleo de Greengrass
CloudWatch componente de métricas	Está disponible la versión 1.1.0 del componente de CloudWatch métricas. Nuevas características • Agregar compatibilidad de recetas para la versión lite del núcleo de Greengrass

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.13.0 el 26 de agosto de 2024

Este lanzamiento proporciona la versión 2.13.0 del componente de núcleo de Greengrass.

Fecha de lanzamiento: 26 de agosto de 2024

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.13.0 del núcleo de Greengrass. Nuevas características • Compatibilidad con puntos de conexión de FIPS en el núcleo. Para obtener más información, consulte Puntos de conexión de FIPS.

Componente	Detalles
	Mejoras y correcciones de errores • Mejoras relacionadas con la cancelación de la implementación: ahora, las implementaciones se pueden cancelar mientras se combina una nueva configuración y mientras se espera a que se inicien los servicios.
Administrador de flujos	Está disponible la versión 2.1.13 del componente administrador de flujos. Mejoras y correcciones de errores • Support FIPS Endpoint en AWS IoT SiteWise
Administrador de secretos	Está disponible la versión 2.2.0 del componente administrador de secretos. Nuevas características • Suma compatibilidad con la actualización periódica de los secretos configurados mediante una nueva clave de configuración del component e. • Suma compatibilidad con un nuevo parámetro de solicitud en la solicitud GetSecretValue de IPC para actualizar los secretos por solicitud
Detector de IP	Está disponible la versión 2.2.0 del componente detector de IP. Nuevas características • Añade compatibilidad con. IPv6 Ahora se puede utilizar IPv6 para mensajería local.
Autenticación del dispositivo de cliente	Está disponible la versión 2.5.1 de la autenticación del dispositivo de cliente. Mejoras y correcciones de errores • Errores y correcciones generales. • Compatible con puntos de conexión de FIPS.

Componente	Detalles
Consola de depuración local	Está disponible la versión 2.4.3 de la consola de depuración local. Mejoras y correcciones de errores • Corrige un problema que mostraba incorrectamente STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH en Mpbs en lugar de bytes/seg.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.6 el 24 de mayo de 2024

Este lanzamiento ofrece la versión 2.12.6 del componente de núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 24 de mayo de 2024

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando

cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.12.6 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema que provocaba un bloqueo al arrancar algunos ARMv8 procesadores, incluido el Jetson Nano.
CLI de Greengrass	Está disponible la versión 2.12.6 de la CLI de Greengrass. Mejoras y correcciones de errores • Versión actualizada de la versión 2.12.6 del núcleo de Greengrass.
Administrador de secretos	Está disponible la versión 2.1.8 del administrador secretos. Mejoras y correcciones de errores • Soluciona un problema por el que el administrador de secretos no aceptaba un ARN parcial.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.5 el 25 de abril de 2024

Este lanzamiento ofrece la versión 2.12.5 del componente de núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 25 de abril de 2024

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.12.5 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona el problema que provocaba que, en ocasiones, la reversión de la implementación se bloqueara al revertir un componente previamente dañado con dependencias sólidas. • Soluciona el problema por el que el núcleo no publicaba actualizaciones de estado tras el aprovisionamiento de la flota. • Agrega reintentos a la API de `GetDeploymentConfiguration` después de recibir errores 404.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.4 el 2 de abril de 2024

Este lanzamiento ofrece la versión 2.12.4 del componente de núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 2 de abril de 2024

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.12.4 del núcleo de Greengrass.

Componente	Detalles
	Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo entra en un punto muerto durante el startup en algunos dispositivos de Linux.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.3 el 27 de marzo de 2024

Este lanzamiento ofrece la versión 2.12.3 del componente de núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 27 de marzo de 2024

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

Important

Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.12.3 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema que provocaba que el núcleo no mostrara el estado correcto de los componentes después del relanzamiento del núcleo y durante la recuperación del componente. • Corrección de errores y mejoras generales.
Administrador de sombras	Está disponible la versión 2.3.7 del componente administrador de sombras. Mejoras y correcciones de errores Soluciona un problema por el que el administrador de sombras registraba periódicamente un error `NullPointerException` durante la sincroniz ación del administrador de sombras.
Aprovisio namiento de flotas	Está disponible la versión 1.2.1 del complemento de aprovisionamiento de flotas de AWS IoT. Mejoras y correcciones de errores Soluciona un problema por el que el complemento de aprovisionamiento de flotas estaba fuera de línea durante el inicio del núcleo de Greengrass. El complemento de aprovisionamiento de flotas ahora reintenta indefinid amente las llamadas de conexión MQTT.
Detector de IP	Está disponible la versión 2.1.9 del componente del spooler de disco Mejoras y correcciones de errores Ajusta el paso de IP adquirido para enviar solo los registros a nivel del registro de depuración.
Component e de agente MQTT 3.1.1 de Moquette	Está disponible la versión 2.3.6 del componente de agente MQTT 3.1.1 de Moquette.

Componente	Detalles
	Mejoras y correcciones de errores Corrección de errores y mejoras generales.
Administrador de Lambda	Está disponible la versión 2.3.3 del componente administrador de Lambda. Mejoras y correcciones de errores Corrección de errores y mejoras generales.
Consola de depuración local	Está disponible la versión 2.4.2 del componente de la consola de depuración local. Mejoras y correcciones de errores Corrección de errores y mejoras generales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.2 el 15 de febrero de 2024

Este lanzamiento ofrece la versión 2.12.2 del componente del núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 15 de febrero de 2024

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que

las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.12.2 del núcleo de Greengrass. Mejoras y correcciones de errores • Corrige un problema por el que los registros antiguos no se eliminaban correctamente. • Corrección de errores y mejoras generales.
Administrador de sombras	Está disponible la versión 2.3.6 del componente administrador de sombras. Mejoras y correcciones de errores Corrige un problema por el que las propiedades de sombra que se eliminan mediante las actualizaciones de Nube de AWS mientras el dispositivo no está en línea siguen existiendo en la sombra local tras recuperar la conectividad.
Lanzador de Lambda	Ya está disponible la versión 2.0.13 del componente lanzador de Lambda. Mejoras y correcciones de errores Corrección de errores y mejoras generales.
Spooler de disco	Está disponible la versión 1.0.3 del componente del spooler de disco.

Componente	Detalles
	Mejoras y correcciones de errores Mejora el rendimiento al reutilizar las conexiones de la base de datos.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.1 el 8 de diciembre de 2023

Este lanzamiento incluye la versión 2.12.1 del componente del núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 8 de diciembre de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

Important

Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.12.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo podía duplicar las suscripciones de MQTT en temas de implementación, lo que provoca más registros y publicaciones en MQTT adicionales.
Autenticación del dispositivo de cliente	Está disponible la versión 2.4.5 del componente de autenticación del dispositivo de cliente. Nuevas características Suma compatibilidad con prefijos comodín para seleccionar los nombres de objetos con el parámetro `selectionRule` . Mejoras y correcciones de errores Soluciona un problema que provocaba que, en algunos casos, los certificados no se actualizarán con nueva información de conectividad.
Spooler de disco	Está disponible la versión 1.0.2 del componente del spooler de disco. Mejoras y correcciones de errores Soluciona un problema por el que el campo de formato de mensaje MQTT no se conserva en algunos casos.
Puente MQTT	Está disponible la versión 2.3.1 del componente del spooler de disco. Mejoras y correcciones de errores Soluciona un problema por el que el cliente MQTT local entra en un bucle de desconexión.
Administrador de flujos	Está disponible la versión 2.1.12 del componente administrador de flujos.

Componente	Detalles
	Mejoras y correcciones de errores Actualiza el orden en que se utilizan las credenciales, de modo que las credenciales de Greengrass son las preferidas para las solicitudes de AWS servicio.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.12.0 el 7 de noviembre de 2023

Esta versión incluye la versión 2.12.0 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados. AWS

Fecha de lanzamiento: 7 de noviembre de 2023

Aspectos destacados del lanzamiento

- Bootstrap en fase de reversión: AWS IoT Greengrass ahora proporciona un parámetro de configuración del núcleo de Greengrass llamado. `BootstrapOnRollback` Esta característica le permite ejecutar los pasos del ciclo de vida del arranque como parte de una implementación de reversión.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS que incluye funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se

implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Esta disponible la versión 2.12.0 del núcleo de Greengrass. Nuevas características • Le permite ejecutar los pasos del ciclo de vida del arranque como parte de una implementación de reversión.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.3 el 18 de octubre de 2023

Este lanzamiento incluye la versión 2.11.3 del componente de núcleo de Greengrass.

Fecha de lanzamiento: 18 de octubre de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.11.3 del núcleo de Greengrass. Mejoras y correcciones de errores • Corrige un problema en el núcleo que puede iniciar incorrectamente un componente cuando fallan sus dependencias. Nuevas características • Agrega un tipo de punto de conexión de S3 configurable.
Administrador de Lambda	Está disponible la versión 2.3.1 del componente Administrador de Lambda. Mejoras y correcciones de errores • Ajusta los niveles de registro para detectar determinados errores.
Consola de depuración local	Está disponible la versión 2.4.0 del componente Administrador de Lambda. Nuevas características • Agrega la consola de depuración del Administrador de flujos.

Componente	Detalles
Administrador de registros	Está disponible la versión 2.3.6 del componente Administrador de registros. Mejoras y correcciones de errores • Ajusta los niveles de registro para detectar determinados errores.
Administrador de sombras	Está disponible la versión 2.3.4 del componente Administrador de sombras. Mejoras y correcciones de errores • Suma compatibilidad con documentos de estado de sombra nulos y vacíos.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.2 el 9 de agosto de 2023

Esta versión proporciona la versión 2.11.2 del componente del núcleo de Greengrass.

Fecha de lanzamiento: 9 de agosto de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.

Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.11.2 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema en el cliente MQTT 5 del núcleo, que podía aparecer desconectado cuando se utilizaban un gran número de suscripciones (> 50). • Agrega un reintento por el error TCP del docker dial.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.1 el 21 de julio de 2023

Este lanzamiento incluye la versión 2.11.1 del componente del núcleo de Greengrass.

Fecha de lanzamiento: 21 de julio de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

Important

Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que

las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.11.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo no se inicia si se produce un error en una tarea de arranque y el archivo de metadatos de implement ación esta dañado. • Soluciona el problema por el que los componentes de Lambda bajo demanda no se incluyen en las actualizaciones del estado de la implementación. • Añade compatibilidad con la política de autorizaciones duplicadas IDs.
Administrador de Lambda	Está disponible la versión 2.2.11 del administrador de Lambda. Mejoras y correcciones de errores • Soluciona un problema por el que la LegacySubscriptionRouter configuración no se actualiza cuando cambia la configuración de Lambda.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.11.0 el 28 de junio de 2023

Este lanzamiento proporciona la versión 2.11.0 del componente del núcleo de Greengrass.

Fecha de lanzamiento: 28 de junio de 2023

Aspectos destacados del lanzamiento

- Spooler de disco persistente: AWS IoT Greengrass ahora proporciona una implementación de spooler persistente para los mensajes enviados desde los dispositivos principales de Greengrass a. AWS IoT Core Este componente almacenará estos mensajes salientes en el disco. Para obtener más información, consulte Spooler de disco.
- Mejoras en la implementación local: ahora puede cancelar las implementaciones locales, establecer políticas de gestión de los problemas de implementación y obtener información detallada sobre el estado de la implementación.
- Mejoras en la velocidad de registro: se han mejorado las velocidades de carga de registros para el componente de administrador de registros.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes que se proporcionan y AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.

Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.11.0 del núcleo de Greengrass. Nuevas características • Le permite cancelar una implementación local. • Le permite configurar una política de gestión de errores para una implementación local. • Suma compatibilidad con un complemento del spooler de disco.
CLI de Greengrass	Está disponible la versión 2.11.0 de la CLI de Greengrass. Nuevas características • Le permite cancelar una implementación local. • Le permite configurar una política de gestión de errores para una implementación local. • Mejora los informes detallados del estado de la implementación.
Spooler de disco	Está disponible la versión 1.0.0 del componente del spooler de disco. • El componente disk spooler proporciona un almacenamiento persistente de los mensajes enviados desde los dispositivos principales de Greengrass a. AWS IoT Core
Administrador de registros	Está disponible la versión 2.3.5 del componente administrador de registros. Mejoras Mejora la velocidad de carga de los registros.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.3 el 21 de junio de 2023

Este lanzamiento incluye la versión 2.10.3 del componente del núcleo de Greengrass.

Fecha de lanzamiento: 21 de junio de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.10.3 del núcleo de Greengrass.

Componente	Detalles
	Mejoras y correcciones de errores • Soluciona un problema por el que Greengrass no se suscribe a las notificaciones de implementación cuando utiliza el proveedor PKCS #11.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.2 el 5 de junio de 2023

Este lanzamiento proporciona la versión 2.10.2 del componente de núcleo de Greengrass.

Fecha de lanzamiento: 5 de junio de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.10.2 del núcleo de Greengrass. Mejoras y correcciones de errores • Permite analizar los ciclos de vida de los componentes sin distinguir entre mayúsculas y minúsculas. • Soluciona el problema por el que la variable PATH del entorno no se recreaba correctamente. • Corrige la codificación URI del proxy para los componentes, incluido el administrador de flujos para los nombres de usuario con caracteres especiales.
Autenticación del dispositivo de cliente	Está disponible la versión 2.4.2 del componente de autenticación del dispositivo de cliente. Nuevas características Agrega una nueva opción de configuración `startupTimeoutSeconds` .
Administrador de Lambda	Está disponible la versión 2.2.9 del componente Administrador de Lambda. Mejoras y correcciones de errores Soluciona un problema por el que el número de puerto estaba dañado debido a un reloj sesgado.
Administrador de registros	Está disponible la versión 2.3.4 del componente administrador de registros. Mejoras y correcciones de errores • Suma soporte para configurar el parámetro `periodicUploadIntervalSec` en valores fraccionarios. El mínimo es 1 microsegundo. • Soluciona un problema por el que el administrador de registros no respetaba los CloudWatch `putLogEvents` límites.
Agente MQTT 3.1 (Moquette)	Está disponible la versión 2.3.3 del componente Agente MQTT 3.1 (Moquette).

Componente	Detalles
	Nuevas características Agrega una nueva opción de configuración `startupTimeoutSeconds` .
Puente MQTT	Está disponible la versión 2.2.6 del componente puente MQTT. Nuevas características Agrega una nueva opción de configuración `startupTimeoutSeconds` .
Administrador de flujos	Está disponible la versión 2.1.7 del componente administrador de flujos. Mejoras y correcciones de errores Soluciona un problema por el que el administrador de flujos no podía leer correctamente la configuración del proxy.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.1 el 11 de mayo de 2023

Este lanzamiento incluye la versión 2.10.1 del componente del núcleo de Greengrass.

Fecha de lanzamiento: 11 de mayo de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

Important

Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.10.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema que podía provocar un bloqueo al arrancar algunos ARMv8 procesadores, incluido el Jetson Nano. • Greengrass ya no cierra el estándar de un componente, lo que revierte el comportamiento al de la versión anterior a la 2.10.0
Administrador de flujos	Está disponible la versión 2.1.6 del nuevo administrador de flujos. Mejoras y correcciones de errores Soluciona un problema que podía provocar un bloqueo al arrancar algunos ARMv8 procesadores, incluido el Jetson Nano.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.10.0 el 9 de mayo de 2023

Este lanzamiento incluye la versión 2.10.0 del componente del núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 9 de mayo de 2023

Aspectos destacados del lanzamiento

- MQTT5 soporte: AWS IoT Greengrass ahora admite el envío y la recepción de mensajes desde el uso. AWS IoT Core MQTT5 Para obtener más información, consulte Publicar mensajes AWS IoT Core MQTT.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por AWS él que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
>
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Esta disponible la versión 2.10.0 del núcleo de Greengrass. Nuevas características • Suma compatibilidad de `interpolateComponentConfiguration` con la expresión regular vacía. Greengrass ahora interpola desde el objeto de configuración raíz. • Añade compatibilidad con MQTT5. • Agrega un mecanismo para cargar los componentes del complemento rápidamente sin necesidad de escanearlos. • Permite a Greengrass ahorrar espacio en disco al eliminar las imágenes de Docker no utilizadas. Mejoras y correcciones de errores • Soluciona un problema por el cual la reversión dejaba ciertos valores de configuración de una implementación en su lugar. • Soluciona un problema por el que el núcleo de Greengrass valida una secuencia de AWS dominios en puntos finales de AWS datos y sin credenciales personalizados. • Actualiza la resolución de dependencias multigrupo para volver a resolver todas las dependencias de los grupos mediante la Nube de AWS negociación, en lugar de limitarlas a la versión activa. Esta actualización también elimina el código de error de implementación `INSTALLED_COMPONENT_NOT_FOUND` . • Actualiza el núcleo de Greengrass para omitir la descarga de imágenes de Docker cuando ya existen localmente. • Actualiza el núcleo de Greengrass para reiniciar el paso de instalación de un componente antes de que se agote el tiempo de espera. • Correcciones y mejoras menores adicionales.
Administrador de sombras	Ya está disponible la versión 2.3.2 del nuevo Administrador de sombras.

Componente	Detalles
	Mejoras y correcciones de errores Soluciona un problema por el que el administrador de sombras entra en estado `BROKEN` cuando la base de datos de sombra local está dañada.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.6 el 20 de abril de 2023

Este lanzamiento incluye la versión 2.9.6 del componente de núcleo de Greengrass.

Fecha de lanzamiento: 20 de abril de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

Important

Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.6 del núcleo de Greengrass. Mejoras y correcciones de errores • Corrige un problema por el que una implementación de Greengras s fallaba con el error LAUNCH_DIRECTORY_CORRUPTED y un posterior reinicio del dispositivo no podía iniciar Greengrass. Este error puede producirse al mover el dispositivo de Greengrass entre varios grupos de elementos con implementaciones que requieren el reinicio de Greengrass.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.5 el 30 de marzo de 2023

Este lanzamiento contiene la versión 2.9.5 del componente de núcleo de Greengrass.

Fecha de lanzamiento: 30 de marzo de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.

Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.5 del núcleo de Greengrass. Nuevas características • Suma compatibilidad con la verificación de firmas del software de núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema por el que una implementación falla cuando la región de metadatos de la receta local no coincide con la región de lanzamiento del núcleo de Greengrass. El núcleo de Greengrass ahora renegocia con la nube cuando esto ocurre. • Soluciona un problema por el que el spooler de mensajes de MQTT se llena y nunca elimina los mensajes. • Correcciones y mejoras menores adicionales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.4 el 24 de febrero de 2023

Este lanzamiento proporciona la versión 2.9.4 del componente de núcleo de Greengrass.

Fecha de lanzamiento: 24 de febrero de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.4 del núcleo de Greengrass. Mejoras y correcciones de errores • Comprueba si hay un mensaje nulo antes de eliminar los mensajes de QOS 0. • Trunca los valores de detalle del estado del trabajo si superan el límite de 1024 caracteres. • Actualiza el script de arranque para Windows para que lea correctam ente la ruta raíz de Greengrass si esa ruta incluye espacios. • Actualiza la suscripción AWS IoT Core para eliminar los mensajes de los clientes si no se envió la respuesta de la suscripción.

Componente	Detalles
	• Garantiza que el núcleo cargue la configuración a partir de archivos de respaldo cuando el archivo de configuración principal esté dañado o falte.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.3 el 1 de febrero de 2023

Este lanzamiento incluye la versión 2.9.3 del componente núcleo de Greengrass.

Fecha de lanzamiento: 1 de febrero de 2023

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes proporcionados por él AWS que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.3 del núcleo de Greengrass. Mejoras y correcciones de errores • Garantiza que los clientes MQTT IDs no estén duplicados. • Agrega una lectura y escritura de archivos más robustas para evitar la corrupción y recuperarse de ella. • Reintenta extraer la imagen de Docker en caso de errores específicos relacionados con la red. • Agrega la opción `noProxyAddresses` de conexión MQTT.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.2 el 22 de diciembre de 2022

Este lanzamiento incluye la versión 2.9.2 del componente núcleo de Greengrass.

Fecha de lanzamiento: 22 de diciembre de 2022

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas

actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.2 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona el problema por el que la configuración de `interpolateComponentConfiguration` no se aplicaba a una implementación en curso. • Utiliza OSHI para enumerar todos los procesos secundarios.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.1 el 18 de noviembre de 2022

Esta versión incluye la versión 2.9.1 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 18 de noviembre de 2022

Aspectos destacados del lanzamiento

- Administrador de registros: el administrador de registros ahora procesa y carga directamente los archivos de registro activos en lugar de esperar a que se roten los archivos nuevos. Esta mejora reduce considerablemente los retrasos en los registros. Para obtener más información, consulte Administrador de registros.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Agrega una corrección por la que Greengrass se reinicia si una implementación elimina un componente del complemento.
Administrador de registros	Está disponible la versión 2.3.0 del nuevo administrador de registros. **Note** Recomendamos que actualice el núcleo de Greengrass a 2.9.1 cuando actualice el administrador de registros a 2.3.0.

Componente	Detalles
	Nuevas características • Reducción de las demoras en el registro cuando se procesan y cargan directamente los archivos de registro activos en lugar de esperar a que se roten los archivos nuevos. Mejoras y correcciones de errores • Mejora de la compatibilidad con la rotación de registros cuando se rotan archivos con un nombre único. • Correcciones y mejoras menores adicionales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.9.0 el 15 de noviembre de 2022

Este lanzamiento incluye la versión 2.9.0 del componente núcleo de Greengrass y actualizaciones de los componentes proporcionados por AWS.

Fecha de lanzamiento: 15 de noviembre de 2022

Aspectos destacados del lanzamiento

- Autenticación sin conexión: AWS IoT Greengrass ahora es compatible con la autenticación sin conexión. Puede configurar su dispositivo AWS IoT Greengrass principal para que los dispositivos cliente puedan conectarse a un dispositivo principal, incluso cuando el dispositivo principal no esté conectado a la nube. Para obtener más información, consulte Autenticación sin conexión.
- Implementaciones secundarias: ahora puede crear implementaciones secundarias. Puede usar una implementación secundaria para resolver las implementaciones fallidas. Cada implementación secundaria puede probar una configuración diferente de una implementación fallida en un subconjunto de dispositivos más pequeño. Para obtener más información, consulte Crear implementaciones secundarias.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.9.0 del núcleo de Greengrass. Nuevas características • Agrega la posibilidad de crear subimplementaciones que reintenten las implementaciones con un subconjunto de dispositivos más pequeño. Esta característica crea una forma más eficiente de probar y resolver las implementaciones fallidas. Mejoras y correcciones de errores • Mejora la compatibilidad con los sistemas que no tienen `useradd`, `groupadd` y `usermod`. • Correcciones y mejoras menores adicionales.

Componente	Detalles
Autenticación del dispositivo de cliente	Está disponible la versión 2.3.0 del componente de autenticación del dispositivo de cliente. Nuevas características • Agrega compatibilidad con la autenticación sin conexión de los dispositivos de cliente. Con esta característica, los dispositivos de cliente pueden seguir conectándose al dispositivo principal cuando este no esté conectado a Internet. • Agrega compatibilidad con las autoridades de certificación (CA) proporcionadas por el cliente. Su dispositivo principal utiliza una CA proporcionada por el cliente como certificado raíz para generar los certificados de agente de MQTT.
Agente MQTT 5 (EMQX)	Está disponible la versión 1.2.0 del componente Agente MQTT 5 (EMQX). Nuevas características Se agregó compatibilidad con las cadenas de certificados.
Agente MQTT de Moquette	Está disponible la versión 2.3.0 del nuevo componente de agente MQTT de Moquette. Nuevas características Se agregó compatibilidad con las cadenas de certificados.
Administrador de secretos	Ya está disponible la versión 2.1.4 del nuevo administrador de secretos. Mejoras y correcciones de errores Corrige un problema en el que los secretos almacenados en caché se eliminaban cuando se implementaba el administrador de secretos y se reiniciaba el núcleo de Greengrass.

Componente	Detalles
Administrador de flujos	Está disponible la versión 2.1.2 del nuevo administrador de flujos. Mejoras y correcciones de errores Corrige un problema en el que el sistema operativo Windows utilizaba un idioma distinto del inglés.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.8.1 el 13 de octubre de 2022

Este lanzamiento incluye la versión 2.8.1 del componente núcleo de Greengrass.

Fecha de lanzamiento: 13 de octubre de 2022

> **Note**
>
> Si utiliza la versión 2.8.0 del núcleo de Greengrass, recomendamos encarecidamente que la actualice a la versión 2.8.1 del núcleo de Greengrass.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos

a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.8.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona el problema por el que los códigos de error de implement ación no se generaban correctamente a partir de errores de la API de Greengrass. • Soluciona el problema que provocaba que las actualizaciones del estado de la flota enviaran información inexacta cuando un component e alcanzaba un estado `ERRORED` determinado durante una implement ación. • Soluciona el problema por el que las implementaciones no se podían completar cuando Greengrass tenía más de 50 suscripciones existente s.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.8.0 el 7 de octubre de 2022

Este lanzamiento incluye la versión 2.8.0 del componente núcleo de Greengrass y la versión 1.1.0 del componente agente de MQTT 5 (EMQX).

Fecha de lanzamiento: 7 de octubre de 2022

Aspectos destacados del lanzamiento

- Códigos de error de la implementación: el núcleo de Greengrass ahora informa de una respuesta del estado de la implementación que incluye códigos de error detallados cuando no se puede completar la implementación de un componente. Para obtener más información, consulte Códigos de error de implementación detallados.
- Estados de error de los componentes: el núcleo de Greengrass ahora informa una respuesta del estado del componente que incluye estados de error detallados cuando un componente entra en el estado `BROKEN` o `ERRORED`. Para obtener más información, consulte Códigos de estado de componentes detallados.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
>
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.8.0 del núcleo de Greengrass. Nuevas características • Actualiza el núcleo de Greengrass para informar de una respuesta sobre el estado de la implementación que incluye códigos de error detallados cuando se produce un problema al implementar los componentes en un dispositivo principal. Para obtener más información, consulte Códigos de error de implementación detallados. • Actualiza el núcleo de Greengrass para informar de una respuesta al estado de un componente que incluye códigos de error detallados cuando un componente entra en el estado `BROKEN` o `ERRORED`. Para obtener más información, consulte Códigos de estado de componentes detallados. • Amplía los campos de los mensajes de estado para mejorar la informaci ón de disponibilidad de los dispositivos en la nube. • Mejora la solidez del servicio de estado de la flota. Mejoras y correcciones de errores • Permite volver a instalar un componente dañado cuando cambia su configuración. • Soluciona el problema por el que el reinicio del núcleo durante la implementación de arranque provoca un error en la implementación. • Soluciona el problema en Windows por el que la instalación falla cuando una ruta raíz contiene espacios. • Soluciona el problema por el que un componente que se apagaba durante una implementación utilizaba el script de apagado de la nueva versión. • Varias mejoras de apagado. • Correcciones y mejoras menores adicionales.
Agente MQTT 5 (EMQX)	Está disponible la versión 1.1.0 del componente Agente MQTT 5 (EMQX).

Componente	Detalles
	Nuevas características • Suma compatibilidad con las configuraciones de EMQX, incluidas las opciones de agente y los complementos. Mejoras y correcciones de errores • Actualiza EMQX a la versión 4.4.9.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.7.0 el 28 de julio de 2022

Este lanzamiento incluye la versión 2.7.0 del componente núcleo de Greengrass, la versión 2.1.0 del componente administrador de flujos y la versión 2.2.5 del componente administrador de Lambda.

Fecha de lanzamiento: 28 de julio de 2022

Aspectos destacados del lanzamiento

- Métricas de telemetría de Stream Manager: ahora, Stream Manager envía automáticamente las métricas de telemetría a Amazon EventBridge, para que puedas crear aplicaciones en la nube que supervisen y analicen el volumen de datos que cargan tus dispositivos principales. Para obtener más información, consulte Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass.
- Autoridad de certificación (CA) personalizada: ahora se admiten los certificados de cliente firmados por una CA de certificados personalizada, en la que la CA no esté registrada. AWS IoT Para obtener más información, consulte Use un certificado de dispositivo firmado por una CA privada.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Ya está disponible la versión 2.7.0 del núcleo de Greengrass. Nuevas características • Actualiza el núcleo de Greengrass para enviar actualizaciones de estado a la AWS IoT Greengrass nube cuando el dispositivo principal aplica un despliegue local. • Añade compatibilidad con los certificados de cliente firmados por una autoridad de certificación (CA) personalizada, en la que la CA no esté registrada. AWS IoT Para utilizar esta característica, puede establecer la nueva opción de configuración `greengrassDataPlaneEndpoint` en `iotdata`. Para obtener más información, consulte Use un certificado de dispositivo firmado por una CA privada. Mejoras y correcciones de errores • Soluciona el problema por el que el núcleo de Greengrass retrasa una implementación en determinadas situaciones cuando el núcleo se detiene o se reinicia. El núcleo reanuda la implementación después de que se reinicie.

Componente	Detalles
	• Actualiza el instalador de Greengrass para que respete el argumento `--start` cuando especifica configurar el software como un servicio del sistema. • Actualiza el comportamiento de SubscribeToComponentUpdates para establecer el ID de implementación en los casos en los que el núcleo actualiza un componente. • Correcciones y mejoras menores adicionales.
Administrador de flujos	Ya está disponible la versión 2.1.0 del componente administrador de flujos. Nuevas características • Actualiza este componente para enviar automáticamente las métricas de telemetría a Amazon. EventBridge Para obtener más información, consulte Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass. Esta característica requiere la versión 2.7.0 o posterior del componente núcleo de Greengrass. • Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
Administrador de Lambda	Está disponible la versión 2.2.5 del componente Administrador de Lambda. Nuevas características • Añade compatibilidad con los comodines de temas de MQTT en las fuentes de eventos en las que te suscribes a mensajes locales. publish/subscribe Esta característica requiere la versión 2.6.0 o posterior del componente núcleo de Greengrass. • Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.6.0 el 27 de junio de 2022

Esta versión incluye la versión 2.6.0 del componente núcleo de Greengrass, los AWS nuevos componentes proporcionados y las actualizaciones AWS de los componentes proporcionados.

Fecha de lanzamiento: 27 de junio de 2022

Aspectos destacados del lanzamiento

- Comodines en publish/subscribe los temas locales: ahora puede usar los comodines de MQTT al suscribirse a temas locales. publish/subscribe Para obtener más información, consulte Publicar/suscribir mensajes locales y SubscribeToTopic.
- Compatibilidad con sombras de dispositivos de cliente: ahora puede interactuar con las sombras de los dispositivos de cliente en componentes personalizados y sincronizar las sombras de los dispositivos de cliente con AWS IoT Core. Para obtener más información, consulte Interacción y sincronización con las sombras de dispositivo de cliente.
- Soporte local de MQTT 5 para dispositivos de cliente: ahora puede implementar el agente MQTT 5 de EMQX para utilizar las características de MQTT 5 en la comunicación entre los dispositivos de cliente y un dispositivo principal. Para obtener más información, consulte Agente MQTT 5 (EMQX) y Conexión de dispositivos de cliente a los dispositivos principales.
- Variables de receta en las configuraciones de los componentes: ahora puede utilizar variables de receta específicas en las configuraciones de los componentes. Puede usar estas variables de receta al definir la configuración predeterminada de un componente en una receta o al configurar un componente en una implementación. Para obtener más información, consulte Variables de receta y Uso de variables de receta en las actualizaciones de combinación.
- Comodín en las políticas de autorización de IPC: ahora puede usar el comodín * para que coincida con cualquier combinación de caracteres en las políticas de autorización de comunicación entre procesos (IPC). Este comodín lo habilita a permitir el acceso a varios recursos en una única política de autorización. Para obtener más información, consulte Comodines en las políticas de autorización.
- Operaciones de IPC que administran las implementaciones y los componentes locales: ahora puede desarrollar componentes personalizados que administren las implementaciones locales y ver los detalles de los componentes. Para obtener más información, consulte IPC: administrar las implementaciones y los componentes locales.

- Operaciones de IPC que autentican y autorizan los dispositivos cliente: ahora puede utilizar estas operaciones para crear un componente de agente local personalizado. Para obtener más información, consulte IPC: Autenticar y autorizar dispositivos de cliente.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.6.0 del núcleo de Greengrass. Nuevas características • Añade compatibilidad con los comodines de MQTT al suscribirse a temas locales publish/subscribe . Para obtener más información, consulte Publicar/suscribir mensajes locales y SubscribeToTopic.

Componente	Detalles
	• Suma compatibilidad con variables de receta en las configuraciones de componentes, distintas de la variable de receta *component_dependency_name* `:configuration:` *json_pointer* . Puede usar estas variables de receta al definir un componente `DefaultConfiguration` en una receta o al configurar un componente en una implementación. Para activar esta función, defina la opción de interpolateComponentConfigurationconfiguración en. `true` Para obtener más información, consulte Variables de receta y Uso de variables de receta en las actualizaciones de combinación. • Agrega una compatibilidad total con el carácter comodín * en las políticas de autorización de la comunicación entre procesos (IPC). Ahora puede especificar el carácter * de una cadena de recursos para que coincida con cualquier combinación de caracteres. Para obtener más información, consulte Comodines en las políticas de autorización. • Agrega compatibilidad con componentes personalizados para llamar a las operaciones de IPC que utiliza la CLI de Greengrass. Puede usar estas operaciones de IPC para administrar las implementaciones locales, ver los detalles de los componentes y generar una contraseña que podrá usar para iniciar sesión en la consola de depuración local. Para obtener más información, consulte IPC: administrar las implementaciones y los componentes locales. Mejoras y correcciones de errores • Soluciona el problema por el que los componentes dependientes no reaccionaban cuando sus dependencias principales se reiniciaban o cambiaban de estado en determinadas situaciones. • Mejora los mensajes de error que el dispositivo principal envía al servicio AWS IoT Greengrass en la nube cuando se produce un error en la implementación. • Soluciona el problema por el que el núcleo de Greengrass aplicaba una implementación de un objeto dos veces en determinadas situaciones cuando el núcleo se reiniciaba. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.

Componente	Detalles
Agente MQTT 5 (EMQX)	Está disponible la versión 1.0.0 del nuevo componente de agente MQTT 5 de EMQX. Nuevas características • Agrega compatibilidad con el agente MQTT 5 de EMQX local. Los dispositivos de cliente se pueden conectar a este agente MQTT para comunicarse con un dispositivo principal mediante las características de MQTT 5.
Administrador de sombras	Está disponible la versión 2.2.0 del componente administrador de sombras. Nuevas características • Añade compatibilidad con el servicio paralelo local a través de la publish/subscribe interfaz local. Ahora puede comunicarse con el intermediario de publish/subscribe mensajes local sobre temas ocultos de MQTT para obtener, actualizar y eliminar sombras en el dispositivo principal. Esta característica le permite conectar los dispositivos de cliente al servicio de sombra local mediante el puente MQTT para retransmitir mensajes sobre temas de sombra entre los dispositivos de cliente y la interfaz local de publicación/suscripción. Esta característica requiere la versión 2.6.0 o posterior del componente núcleo de Greengrass. Para conectar los dispositivos de cliente al servicio de sombra local, también debe utilizar la versión 2.2.0 o posterior del componente puente MQTT. • Agrega la opción `direction` que puede configurar para personalizar la dirección de sincronización de las sombras entre el servicio de sombra local y la Nube de AWS. Puede configurar esta opción para reducir el ancho de banda y las conexiones a la Nube de AWS.

Componente	Detalles
Autenticación del dispositivo de cliente	Está disponible la versión 2.2.0 del componente de autenticación del dispositivo de cliente. Nuevas características • Suma compatibilidad con componentes personalizados para llamar a las operaciones de comunicación entre procesos (IPC) a fin de autenticar y autorizar los dispositivos de cliente. Puede usar estas operaciones en un componente de agente MQTT personalizado, por ejemplo. Para obtener más información, consulte IPC: Autenticar y autorizar dispositivos de cliente. • Agrega las opciones `maxActiveAuthTokens` , `cloudQueueSize` y `threadPoolSize` que puede configurar para ajustar el rendimiento de este componente.
Puente MQTT	Está disponible la versión 2.2.0 del componente puente de MQTT. Nuevas características • Añade compatibilidad con los caracteres comodín (#y+) de los temas MQTT cuando se especifica local publish/subscribe como agente de mensajes de origen. Esta característica requiere la versión 2.6.0 o posterior del componente núcleo de Greengrass. • Agrega la opción `targetTopicPrefix` , que puede especificar, para configurar el puente MQTT de forma que agregue un prefijo al tema de destino cuando retransmita un mensaje.

Componente	Detalles
CLI de Greengrass	Está disponible la versión 2.6.0 de la CLI de Greengrass. Nuevas características • Agrega compatibilidad con componentes personalizados para llamar a las operaciones de comunicación entre procesos (IPC) que utiliza la CLI de Greengrass. Puede usar estas operaciones de IPC para administr ar las implementaciones locales, ver los detalles de los component es y generar una contraseña que podrá usar para iniciar sesión en la consola de depuración local. Para obtener más información, consulte IPC: administrar las implementaciones y los componentes locales. Mejoras y correcciones de errores • Correcciones y mejoras menores adicionales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.6 el 31 de mayo de 2022

Este lanzamiento incluye la versión 2.5.6 del componente de núcleo de Greengrass y la versión 2.2.4 del componente administrador de registros.

Fecha de lanzamiento: 31 de mayo de 2022

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se

implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.6 del núcleo de Greengrass. Nuevas características • Suma compatibilidad con los módulos de seguridad de hardware que utilizan claves ECC. Puede utilizar un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado del dispositivo. Para obtener más información, consulte Integración de la seguridad de hardware. Mejoras y correcciones de errores • Soluciona el problema por el que la implementación nunca se completa cuando se implementa un componente con un script de instalación defectuoso en determinadas situaciones. • Mejora el rendimiento durante el arranque. • Correcciones y mejoras menores adicionales.
Administrador de registros	Está disponible la versión 2.2.4 del componente administrador de registros. Mejoras y correcciones de errores • Mejora de la estabilidad cuando se gestionan configuraciones no válidas. • Correcciones y mejoras menores adicionales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.5 el 6 de abril de 2022

Este lanzamiento contiene la versión 2.5.5 del componente núcleo de Greengrass.

Fecha de lanzamiento: 6 de abril de 2022

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.5 del núcleo de Greengrass.

Componente	Detalles
	Nuevas características • Agrega la variable de entorno `GG_ROOT_CA_PATH` para los componentes, de forma que pueda acceder al certificado raíz de la autoridad de certificados (CA) raíz en los componentes personalizados. Mejoras y correcciones de errores • Suma compatibilidad con dispositivos Windows que utilizan un idioma de visualización distinto del inglés. • Actualiza la forma en que el núcleo de Greengrass analiza los argumentos booleanos del instalador, de modo que puede especificar un argumento booleano sin un valor booleano para especificar un valor `true`. Por ejemplo, ahora puede especificar `--provision` en lugar de `--provision true` para instalar con el aprovisionamiento automático de recursos. • Soluciona el problema por el que el dispositivo principal no informaba de su estado al servicio en la nube de AWS IoT Greengrass después del aprovisionamiento en determinadas situaciones. • Correcciones y mejoras menores adicionales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.4 el 23 de marzo de 2022

Esta versión incluye la versión 2.5.4 del componente núcleo de Greengrass y la versión 2.0.10 del componente lanzador de Lambda.

Fecha de lanzamiento: 23 de marzo de 2022

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.4 del núcleo de Greengrass. Mejoras y correcciones de errores • Corrección de errores y mejoras generales.
Lanzador de Lambda	Está disponible la versión 2.0.10 del componente lanzador de Lambda. Mejoras y correcciones de errores • Corrección de errores y mejoras generales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.3 el 6 de enero de 2022

Esta versión incluye la versión 2.5.3 del componente núcleo de Greengrass y el nuevo componente proveedor PKCS #11.

Fecha de lanzamiento: 6 de enero de 2022

Aspectos destacados del lanzamiento

- Integración de la seguridad del hardware: ahora puede configurar el software AWS IoT Greengrass Core para que utilice una clave privada y un certificado que se almacenarán de forma segura en un módulo de seguridad de hardware (HSM). Para obtener más información, consulte Integración de la seguridad de hardware.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
>
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.3 del núcleo de Greengrass.

Componente	Detalles
	Nuevas características • Agrega compatibilidad con la integración de seguridad de hardware. Puede utilizar un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado del dispositivo. Para obtener más información, consulte Integración de la seguridad de hardware. Mejoras y correcciones de errores • Soluciona el problema con las excepciones de tiempo de ejecución mientras el núcleo establece conexiones MQTT con AWS IoT Core.
Proveedor PKCS#11	Está disponible la versión 2.0.0 del componente de proveedor PKCS #11. Nuevas características • Agrega compatibilidad con la integración de seguridad de hardware. Puede utilizar un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado del dispositivo. Para obtener más información, consulte Integración de la seguridad de hardware.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.2 el 3 de diciembre de 2021

Este lanzamiento proporciona la versión 2.5.2 del componente núcleo de Greengrass.

Fecha de lanzamiento: 3 de diciembre de 2021

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.2 del núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona el problema por el que, una vez actualizado el núcleo de Greengrass, el servicio de Windows no se puede iniciar de nuevo después de detenerlo o reiniciar el dispositivo.
AWS IoT Device Defender	Está disponible la versión 3.0.1 del componente AWS IoT Device Defender. Esta versión del componente AWS IoT Device Defender espera parámetros de configuración diferentes a los de la versión 2.x. Si usa una configuración no predeterminada para la versión 2.x y desea actualizar de la versión 2.x a la versión 3.x, debe actualizar la configuración del componente. Para obtener más información, consulte configuración del componente AWS IoT Device Defender. Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows.

Componente	Detalles
	• Cambia el tipo de componente, de componente de Lambda a componente genérico. Este componente ya no depende del component e del enrutador de suscripciones antiguo para crear suscripciones. • Agrega el nuevo parámetro de configuración `UseInstaller` que permite deshabilitar opcionalmente el script de instalación que instala las dependencias del componente.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.1 el 23 de noviembre de 2021

Esta versión proporciona la versión 2.5.1 del componente núcleo de Greengrass.

Fecha de lanzamiento: 23 de noviembre de 2021

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de

actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.1 del núcleo de Greengrass. Mejoras y correcciones de errores • Agrega compatibilidad con las versiones de 32 bits del Entorno de ejecución de Java (JRE) en Windows. • Cambia el comportamiento de eliminación de grupos de objetos en los dispositivos principales cuya política AWS IoT no concede el permiso `greengrass:ListThingGroupsForCoreDevice` . Con esta versión, la implementación continúa, registra una advertencia y no elimina componentes al quitar el dispositivo principal de un grupo de objetos. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos. • Soluciona el problema con las variables de entorno del sistema que el núcleo de Greengrass pone a disposición de los procesos de los componentes de Greengrass. Ahora puede reiniciar un componente para que utilice las variables de entorno del sistema más recientes.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.5.0 el 12 de noviembre de 2021

Esta versión incluye la versión 2.5.0 del componente núcleo de Greengrass, los AWS nuevos componentes proporcionados y las actualizaciones AWS de los componentes proporcionados.

Fecha de lanzamiento: 12 de noviembre de 2021

Aspectos destacados del lanzamiento

- Compatibilidad con dispositivos Windows: ahora puede ejecutar el software AWS IoT Greengrass principal en dispositivos con sistemas operativos Windows. Para obtener más información, consulte Compatibilidad de características de Greengrass.

- Nuevo comportamiento de eliminación de grupos de objetos: ahora puede eliminar un dispositivo principal de un grupo de objetos para eliminar los componentes de ese grupo en la próxima implementación en ese dispositivo.

  > **Important**
  >
  > Como resultado de este cambio, la AWS IoT política de un dispositivo principal debe tener el `greengrass:ListThingGroupsForCoreDevice` permiso. Si usó el instalador de software AWS IoT Greengrass Core para aprovisionar recursos, la AWS IoT política predeterminada lo permite`greengrass:*`, e incluye este permiso. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

- Soporte de seguridad de hardware: ahora puede configurar el software AWS IoT Greengrass Core para que utilice un módulo de seguridad de hardware (HSM), de forma que pueda almacenar de forma segura la clave privada y el certificado del dispositivo. Para obtener más información, consulte Integración de la seguridad de hardware.
- Compatibilidad con el proxy HTTPS: ahora puede configurar el software AWS IoT Greengrass Core para que se conecte a través de proxies HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Detalles del lanzamiento

- Actualizaciones de compatibilidad con plataformas
- Actualizaciones de componentes públicos

Actualizaciones de compatibilidad con plataformas

Plataforma	Detalles
Windows	AWS IoT Greengrass ahora admite la ejecución del software AWS IoT Greengrass principal en las siguientes versiones de Windows: • Windows 10 • Windows Server 2019

Plataforma	Detalles
	Para obtener más información, consulte Compatibilidad de características de Greengrass.

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.5.0 del núcleo de Greengrass. Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Modifica el comportamiento de la eliminación de grupos de objetos. Con esta versión, puede eliminar un dispositivo principal de un grupo de

Componente	Detalles
	objetos para desinstalar los componentes de ese grupo de objetos en la siguiente implementación. Como resultado de este cambio, la AWS IoT política de un dispositivo principal debe contar con el `greengrass:ListThingGroupsForCoreDevice` permiso. Si usó el instalador de software AWS IoT Greengrass Core para aprovisionar recursos, la AWS IoT política predeterminada lo permite `greengrass:*` , e incluye este permiso. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass. • Suma compatibilidad con las configuraciones de proxy HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red. • Suma el nuevo parámetro de configuración `windowsUser` . Puede usar este parámetro para especificar el usuario predeterminado que se utilizará para ejecutar los componentes en un dispositivo principal de Windows. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes. • Agrega las nuevas opciones de configuración `httpClient` que puede utilizar para personalizar los tiempos de espera de las solicitudes HTTP a fin de mejorar el rendimiento en redes lentas. Para obtener más información, consulte el parámetro de configuración httpClient. Mejoras y correcciones de errores • Corrige la opción de ciclo de vida de arranque para reiniciar el dispositivo principal desde un componente. • Suma compatibilidad con guiones en las variables de la receta. • Corrige la autorización de IPC para los componentes de la función de Lambda bajo demanda. • Mejora los mensajes de registro y cambia los registros no críticos de `INFO` a `DEBUG`, por lo que los registros son más útiles. • Elimina el `iot:DescribeCertificate` permiso de la función de intercambio de fichas predeterminada que el núcleo de Greengrass crea

Componente	Detalles
	al instalar el software AWS IoT Greengrass Core con aprovisionamiento automático. El núcleo de Greengrass no utiliza este permiso. • Soluciona el problema por el que el script de aprovisionamiento automático no requería el permiso `iam:GetPolicy` si `iam:CreatePolicy` estaba disponible para la misma política. • Correcciones y mejoras menores adicionales.
CLI de Greengrass	Está disponible la versión 2.5.0 de la CLI de Greengrass. Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Agrega el nuevo parámetro de configuración `AuthorizedWindowsGroups` que puede especificar para autorizar a los grupos del sistema a utilizar la CLI de Greengrass en dispositivos Windows. • Agrega el parámetro `windowsUser` para las implementaciones locales. Puede usar este parámetro para especificar el usuario que se utilizará para ejecutar los componentes en un dispositivo principal de Windows.

Componente	Detalles
CloudWatch métricas	Está disponible la versión 3.0.0 del componente de CloudWatchmétricas. Esta versión del componente de CloudWatch métricas espera parámetros de configuración diferentes a los de la versión 2.x. Si usa una configuración no predeterminada para la versión 2.x y desea actualizar de la versión 2.x a la versión 3.x, debe actualizar la configuración del componente. Para obtener más información, consulte la configuración del componente de CloudWatch métricas. Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Cambia el tipo de componente, de componente de Lambda a componente genérico. Este componente ya no depende del component e del enrutador de suscripciones antiguo para crear suscripciones. • Agrega un nuevo parámetro de configuración `InputTopic` para especificar el tema al que se suscribe el componente para recibir mensajes. • Agrega un nuevo parámetro de configuración `OutputTopic` para especificar el tema en el que el componente publica las respuestas de estado. • Añade un nuevo parámetro `PubSubToIoTCore` de configuración para especificar si se deben publicar o suscribirse a los temas de AWS IoT Core MQTT. • Agrega el nuevo parámetro de configuración `UseInstaller` que permite deshabilitar opcionalmente el script de instalación que instala las dependencias del componente. Mejoras y correcciones de errores Suma compatibilidad con las marcas de tiempo duplicadas en los datos de entrada.

Componente	Detalles
Administrador de Lambda	Está disponible la versión 2.2.0 del componente administrador de Lambda. Mejoras y correcciones de errores • Soluciona un problema por el que las funciones de Lambda no podían escribir registros luego de un reinicio. • Soluciona un problema por el que el enrutador de suscripciones antiguo enviaba mensajes duplicados cuando había caracteres comodín en el tema. • Soluciona un problema por el que las funciones de Lambda no ancladas no podían utilizar la biblioteca de comunicación entre procesos (IPC) de Greengrass en el SDK para dispositivos con AWS IoT.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.4.0 el 3 de agosto de 2021

Esta versión incluye la versión 2.4.0 del componente núcleo de Greengrass, los AWS nuevos componentes proporcionados y las actualizaciones AWS de los componentes proporcionados.

Fecha de lanzamiento: 3 de agosto de 2021

Aspectos destacados del lanzamiento

- Límites de recursos del sistema: el componente núcleo de Greengrass ahora admite los límites de recursos del sistema. Puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente pueden usar en el dispositivo principal. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes.
- Pausa y reanudación de componentes: el núcleo de Greengrass ahora admite pausar y reanudar componentes. Puede utilizar la biblioteca de comunicación entre procesos (IPC) para desarrollar componentes personalizados que detengan y reanuden los procesos de otros componentes. Para obtener más información, consulte PauseComponent y ResumeComponent.
- Instalación con aprovisionamiento de AWS IoT flotas: utilice el nuevo complemento de aprovisionamiento de AWS IoT flotas para instalar el software AWS IoT Greengrass Core en los dispositivos que se conecten para aprovisionar los recursos necesarios. AWS IoT AWS Los dispositivos utilizan un certificado de reclamación para el aprovisionamiento. Puede incrustar el

certificado de reclamación en los dispositivos durante la fabricación, de modo que cada dispositivo pueda aprovisionarse en cuanto se conecte a Internet. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota.

- Instalación con aprovisionamiento personalizado: desarrolle un complemento de aprovisionamiento personalizado para aprovisionar AWS los recursos necesarios al instalar el software Core en los dispositivos. AWS IoT Greengrass Puede crear una aplicación Java que se ejecute durante la instalación para configurar los dispositivos principales de Greengrass para su caso de uso personalizado. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento de recursos personalizado.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
>
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Ya está disponible la versión 2.4.0 del núcleo de Greengrass. Nuevas características • Suma compatibilidad con los límites de recursos del sistema. Puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente pueden usar en el dispositivo principal. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes. • Agrega operaciones de IPC para pausar y reanudar los component es. Para obtener más información, consulte PauseComponent y ResumeComponent. • Suma compatibilidad con el aprovisionamiento de complementos. Puede especificar un archivo JAR para que se ejecute durante la instalación a fin de aprovisionar AWS los recursos necesarios para un dispositivo principal de Greengrass. El núcleo de Greengrass incluye una interfaz que puede implementar para desarrollar complementos de aprovisionamiento personalizados. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisio namiento de recursos personalizado. • Agrega el `thing-name-policy` argumento opcional al instalador del software AWS IoT Greengrass principal. Puede usar esta opción para especificar una AWS IoT política existente o personalizada al instalar el software AWS IoT Greengrass Core con aprovisionamiento automático de recursos. Mejoras y correcciones de errores • Actualiza la configuración de registro en el arranque. Esto soluciona el problema por el que la configuración de registro no se aplicaba en el arranque. • Actualiza el enlace simbólico del cargador de núcleos para que apunte al almacén de componentes de la carpeta raíz de Greengrass durante la instalación. Esta actualización le permite eliminar el archivo JAR y otros artefactos del núcleo que se descargan al instalar el software AWS IoT Greengrass Core.

Componente	Detalles
	• Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
CLI de Greengrass	Ya está disponible la versión 2.4.0 de la CLI de Greengrass. Nuevas características • Suma compatibilidad con los límites de recursos del sistema. Al crear una implementación local, puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente puede utilizar en el dispositivo principal. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes o el comando create de la implementación.
AWS IoT aprovisionamiento de flota por reclamación	El complemento AWS IoT de aprovisionamiento de flotas mediante reclamación ya está disponible. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota. Nuevas características • Añade soporte para instalar el software AWS IoT Greengrass Core con el aprovisionamiento de AWS IoT flotas. Durante la instalación, los dispositivos se conectan AWS IoT para aprovisionar AWS los recursos necesarios y descargar los certificados de los dispositivos para utilizarlos en las operaciones habituales.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.3.0 el 29 de junio de 2021

Este lanzamiento proporciona la versión 2.3.0 del componente núcleo de Greengrass.

Fecha de lanzamiento: 29 de junio de 2021

Aspectos destacados del lanzamiento

- Compatibilidad con configuraciones de gran tamaño: el componente núcleo de Greengrass ahora admite documentos de implementación de hasta 10 MB. Ahora puede implementar actualizaciones de configuración de mayor tamaño en los componentes de Greengrass.

> **Note**
>
> Para usar esta función, la AWS IoT política del dispositivo principal debe permitir el permiso. `greengrass:GetDeploymentConfiguration` Si utilizaste el instalador del software AWS IoT Greengrass principal para aprovisionar recursos, la AWS IoT política de tu dispositivo principal lo permite`greengrass:*`, e incluye este permiso. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de

actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Ya está disponible la versión 2.3.0 del núcleo de Greengrass. Nuevas características • Agrega compatibilidad con la implementación de documentos de configuración de hasta 10 MB, en lugar de 7 KB (para implementaciones dirigidas a objetos) o 31 KB (para implementaciones dirigidas a grupos de objetos). Para usar esta función, la AWS IoT política del dispositivo principal debe permitir el `greengrass:GetDeploymentConfiguration` permiso. Si utilizaste el instalador del software AWS IoT Greengrass principal para aprovisionar recursos, la AWS IoT política de tu dispositivo principal lo permite `greengrass:*` , e incluye este permiso. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass. • Agrega la variable de la receta `iot:thingName` . Puedes usar esta variable de receta para obtener el nombre del dispositivo AWS IoT principal en una receta. Para obtener más información, consulte Variables de receta. Mejoras y correcciones de errores • Correcciones y mejoras menores adicionales. Para obtener más información, consulta las versiones en GitHub.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.2.0 el 18 de junio de 2021

Esta versión incluye la versión 2.2.0 del componente núcleo de Greengrass, los AWS nuevos componentes proporcionados y las actualizaciones AWS de los componentes proporcionados.

Fecha de lanzamiento: 18 de junio de 2021

Aspectos destacados del lanzamiento

- Compatibilidad con dispositivos cliente: los nuevos componentes AWS de dispositivos cliente proporcionados le permiten conectar los dispositivos cliente a sus dispositivos principales mediante la detección en la nube. Puede sincronizar los dispositivos cliente AWS IoT Core e interactuar con los dispositivos cliente en los componentes de Greengrass. Para obtener más información, consulte Interacción con dispositivos IoT locales.
- Servicio de sombra local: el nuevo componente administrador de sombras habilita el servicio de sombra local en sus dispositivos principales. Puede utilizar este servicio de sombras para interactuar con las sombras locales sin conexión a Internet mediante las bibliotecas de comunicación entre procesos (IPC) de Greengrass en el SDK para dispositivos con AWS IoT. También puede usar el componente administrador de sombras para sincronizar los estados ocultos locales con. AWS IoT Core Para obtener más información, consulte Interacción con las sombras de dispositivo.

Detalles del lanzamiento

- Actualizaciones de componentes públicos

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de

actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Ya está disponible la versión 2.2.0 del núcleo de Greengrass. Nuevas características • Agrega operaciones de IPC para la administración de sombras locales. Mejoras y correcciones de errores • Reduce el tamaño del archivo JAR. • Reduce el uso de la memoria. • Soluciona problemas por los que la configuración del registro no se actualizaba en determinados casos. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
Administrador de sombras	Ya está disponible la versión 2.0.0 del componente administrador de sombras. Nuevas características • Agrega compatibilidad con sombras clásicas y sombras con nombre. • Agrega compatibilidad con la administración de sombras locales mediante IPC. • Añade compatibilidad con la sincronización oculta con AWS IoT Core.
Autenticación del dispositivo de cliente	Ya está disponible la versión 2.0.0 del nuevo componente de autenticación del dispositivo de cliente. Nuevas características • Agrega compatibilidad con dispositivos de cliente de Greengrass, que son dispositivos de IoT locales que se conectan a un dispositivo principal a través de MQTT.

Componente	Detalles
	• Agrega compatibilidad con la autenticación y la autorización de los dispositivos de cliente y sus acciones de MQTT.
Agente MQTT de Moquette	Ya está disponible la versión 2.0.0 del nuevo componente agente MQTT de Moquette. Nuevas características • Agrega compatibilidad con un agente MQTT local de Moquette que se encarga de la comunicación con los dispositivos de cliente.
Puente MQTT	Está disponible la versión 2.0.0 del nuevo componente puente de MQTT. Nuevas características • Añade soporte para retransmitir mensajes entre el bróker MQTT local, el bróker publish/subscribe Greengrass local y AWS IoT Core el bróker MQTT.
Detector de IP	Está disponible la versión 2.0.0 del nuevo componente de detección de IP. Nuevas características • Añade soporte para enviar los puntos finales del intermediario MQTT local de un dispositivo principal al servicio en la AWS IoT Greengrass nube para que puedan conectarse los dispositivos cliente.
Administrador de registros	Ya está disponible la versión 2.1.1 del componente administrador de registros . Mejoras y correcciones de errores • Solución de un problema por el que la configuración del registro del sistema no se actualizaba en algunos casos.
Detección de objetos del DLR	Ya está disponible la versión 2.1.2 de la detección de objetos del DLR. Mejoras y correcciones de errores • Corrige un problema de escalado de la imagen que provocaba que los recuadros delimitadores no fueran precisos en los resultados de la inferencia de detección de objetos del DLR de muestra.

Componente	Detalles
TensorFlow Detección de objetos Lite	Está disponible la versión 2.1.1 de la detección de objetos TensorFlow Lite. Mejoras y correcciones de errores • Corrige un problema de escalado de la imagen que provocaba que los recuadros delimitadores no fueran precisos en los resultados de la inferencia de detección de objetos de TensorFlow Lite de muestra.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.1.0 el 26 de abril de 2021

Este lanzamiento proporciona la versión 2.1.0 del componente núcleo de Greengrass y actualiza los componentes proporcionados por AWS.

Fecha de lanzamiento: 26 de abril de 2021

Aspectos destacados del lanzamiento

- Integración de Docker Hub y Amazon Elastic Container Registry (Amazon ECR): el nuevo componente administrador de aplicaciones de Docker le permite descargar imágenes públicas o privadas de Amazon ECR. También puede usar este componente para descargar imágenes públicas de Docker Hub y AWS Marketplace Para obtener más información, consulte Ejecución de un contenedor de Docker.
- Dockerfile e imágenes de Docker para el software AWS IoT Greengrass Core: puede utilizar la imagen de Docker de Greengrass para ejecutarla en AWS IoT Greengrass un contenedor de Docker que utilice Amazon Linux 2 como sistema operativo base. También puedes usar el AWS IoT Greengrass Dockerfile para crear tu propia imagen de Greengrass. Para obtener más información, consulte Ejecute AWS IoT Greengrass el software principal en un contenedor de Docker.
- Soporte para marcos y plataformas de aprendizaje automático adicionales: puede implementar componentes de inferencia de aprendizaje automático de muestra que utilizan modelos previamente entrenados para realizar la clasificación de imágenes de muestra y la detección de objetos con TensorFlow Lite 2.5.0 y DLR 1.6.0. Esta versión también amplía los ejemplos de compatibilidad con el aprendizaje automático para los dispositivos Armv8 (). AArch64 Para obtener más información, consulte Cómo realizar la inferencia de machine learning.

Detalles del lanzamiento

- Actualizaciones de compatibilidad con plataformas
- Actualizaciones de componentes públicos

Actualizaciones de compatibilidad con plataformas

Plataforma	Detalles
Docker	Ya están disponibles un Dockerfile y una imagen de Docker. AWS IoT Greengrass Dockerfile AWS IoT Greengrass proporciona un Dockerfile para crear una imagen de contenedor que tiene el software AWS IoT Greengrass principal y las dependencias instaladas en una imagen base de Amazon Linux 2 (x86_64). Puede modificar la imagen base del Dockerfile para que se ejecute en una arquitectura de plataforma diferente. AWS IoT Greengrass Imagen de Docker AWS IoT Greengrass proporciona una imagen de Docker prediseñada que tiene el software AWS IoT Greengrass principal y las dependencias instaladas en una imagen base de Amazon Linux 2 (x86_64). Para obtener más información, consulte Ejecute AWS IoT Greengrass el software principal en un contenedor de Docker.

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que

las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles

Núcleo de Greengrass	Ya está disponible la versión 2.1.0 del núcleo de Greengrass.
	
	Nuevas características
	• Admite la descarga de imágenes de Docker desde repositorios privados en Amazon ECR.
	• Agrega los siguientes parámetros para personalizar la configuración de MQTT en los dispositivos principales:
	  • `maxInFlightPublishes` : el número máximo de mensajes QoS 1 de MQTT sin confirmar que pueden estar en proceso al mismo tiempo.
	  • `maxPublishRetry` : el número máximo de reintentos permitidos para un mensaje que no se publica.
	• Agrega el parámetro de configuración `fleetstatusservice` para configurar el intervalo en el que el dispositivo principal publica el estado del dispositivo en la Nube de AWS.
	• Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
	
	Mejoras y correcciones de errores
	• Soluciona el problema que provocaba que las implementaciones de sombras se duplicaran cuando se reiniciaba el núcleo.

Componente	Detalles
	• Soluciona el problema que provocaba que el núcleo se bloqueara cuando detectaba una excepción de carga de servicio. • Mejora la resolución de dependencias de componentes para evitar que se produzca un error en una implementación que incluya una dependencia circular. • Soluciona el problema que impedía volver a implementar un component e de un complemento si este se había eliminado previamente del dispositivo principal. • Se corrigió el problema que provocaba que la variable de entorno `HOME` se estableciera en el directorio *`/greengrass/v2`* `/work` de los componentes de Lambda o de los componentes que se ejecutan como raíz. La variable `HOME` ahora está correctamente configurada en el directorio principal del usuario que ejecuta el componente. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
Administrador de aplicaciones de Docker	Está disponible la versión 2.0.0 del nuevo componente administrador de aplicaciones de Docker. Nuevas características • Administra las credenciales para descargar imágenes de repositorios privados en Amazon ECR. • Descarga imágenes públicas de Amazon ECR, Docker Hub y. AWS Marketplace
Lanzador de Lambda	Está disponible la versión 2.0.4 del componente lanzador de Lambda. Mejoras y correcciones de errores • Solución de un problema por el que el componente `AddGroupOwner` no pasa correctamente al contenedor de función de Lambda.

Componente	Detalles
Enrutador de suscripción antigua	Ya está disponible la versión 2.1.0 del componente enrutador de suscripción antigua. Mejoras y correcciones de errores • Añade soporte para especificar los nombres de los componentes en lugar de ARNs para `source` y. `target` Si especifica el nombre de un componente para una suscripción, no es necesario volver a configurar la suscripción cada vez que cambie la versión de la función de Lambda.
Consola de depuración local	Ya está disponible la versión 2.1.0 del componente de la consola de depuración local. Nuevas características • Utiliza HTTPS para proteger la conexión a la consola de depuración local. HTTPS está habilitado de forma predeterminada. Mejoras y correcciones de errores • Puede descartar los mensajes de la barra flash en el editor de configura ción.

Componente	Detalles
Administrador de registros	Ya está disponible la versión 2.1.0 del componente administrador de registros . Mejoras y correcciones de errores • Utilice valores predeterminados para `logFileDirectoryPath` y `logFileRegex` que funcionen para los componentes de Greengrass que imprimen con salida estándar (stdout) y error estándar (stderr). • Dirija correctamente el tráfico a través de un proxy de red configurado al cargar registros en CloudWatch Logs. • Maneje correctamente los dos puntos (`:`) en los nombres de los flujos de registro. CloudWatch Los nombres de los flujos de registro de registros no admiten signos de dos puntos. • Simplifique los nombres del flujo de registro al eliminar los nombres de los grupos de objetos del flujo de registro. • Elimine un mensaje de registro de errores que se imprime con un comportamiento normal.

Componente	Detalles
Clasificación de imágenes de DLR	Ya está disponible la versión 2.1.1 del componente de clasificación de imágenes de DLR. Nuevas características • Utilice la versión 1.6.0 del Tiempo de ejecución de aprendizaje profundo. • Añada compatibilidad con la clasificación de imágenes de muestra en las plataformas Armv8 ()AArch64. Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano. • Habilite la integración de la cámara para la inferencia de muestras. Utilice el nuevo parámetro de configuración `UseCamera` para permitir que el código de inferencia de muestra acceda a la cámara del dispositivo principal de Greengrass y ejecute la inferencia localmente en la imagen capturada. • Suma compatibilidad para publicar los resultados de la inferencia en la Nube de AWS. Utilice el nuevo parámetro de configuración `PublishResultsOnTopic` para especificar el tema en el que desea publicar los resultados. • Agregue el nuevo parámetro de configuración `ImageDirectory` que le permite especificar un directorio personalizado para la imagen en la que desea realizar la inferencia. Mejoras y correcciones de errores • Escriba los resultados de la inferencia en el archivo de registro del componente en lugar de en un archivo de inferencia independiente. • Utilice el módulo de registro del software AWS IoT Greengrass Core para registrar la salida de los componentes. • Utilice el SDK para dispositivos con AWS IoT para leer la configuración del componente y aplicar los cambios de configuración.

Componente	Detalles
Detección de objetos del DLR	Ya está disponible la versión 2.1.1 del componente de detección de objetos del DLR. Nuevas características • Utilice la versión 1.6.0 del Tiempo de ejecución de aprendizaje profundo. • Agregue soporte para la detección de objetos de muestra en las plataformas Armv8 (AArch64). Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano. • Habilite la integración de la cámara para la inferencia de muestras. Utilice el nuevo parámetro de configuración `UseCamera` para permitir que el código de inferencia de muestra acceda a la cámara del dispositivo principal de Greengrass y ejecute la inferencia localmente en la imagen capturada. • Suma compatibilidad para publicar los resultados de la inferencia en la Nube de AWS. Utilice el nuevo parámetro de configuración `PublishResultsOnTopic` para especificar el tema en el que desea publicar los resultados. • Agregue el nuevo parámetro de configuración `ImageDirectory` que le permite especificar un directorio personalizado para la imagen en la que desea realizar la inferencia. Mejoras y correcciones de errores • Escriba los resultados de la inferencia en el archivo de registro del componente en lugar de en un archivo de inferencia independiente. • Utilice el módulo de registro del software AWS IoT Greengrass Core para registrar la salida de los componentes. • Utilice el SDK para dispositivos con AWS IoT para leer la configuración del componente y aplicar los cambios de configuración.

Componente	Detalles
Almacén de modelos de clasificación de imágenes de DLR	Ya está disponible la versión 2.1.1 del componente almacén de modelos de clasificación de imágenes de DLR. Nuevas características • Agregue un ejemplo de modelo de clasificación de imágenes de ResNet -50 para las plataformas Armv8 ()AArch64. Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano.
Almacén de modelos de detección de objetos del DLR	Ya está disponible la versión 2.1.1 del componente almacén de modelos de detección de objetos del DLR. Nuevas características • Agregue un modelo de detección de YOLOv3 objetos de muestra para las plataformas Armv8 (). AArch64 Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano.
Instalador del DLR	Ya está disponible la versión 1.6.1 del componente DLR. Nuevas características • Instala la versión 1.6.0 del tiempo de ejecución de aprendizaje profundo y sus dependencias. • Añada compatibilidad con la instalación de DLR en las plataformas Armv8 (). AArch64 Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano. Mejoras y correcciones de errores • Instálelo SDK para dispositivos con AWS IoT en el entorno virtual para leer la configuración del componente y aplicar los cambios de configura ción. • Correcciones de errores y mejoras menores adicionales.

Componente	Detalles
TensorFlow Clasificación de imágenes de Lite	Está disponible la versión 2.1.0 del nuevo componente de clasificación de imágenes TensorFlow Lite. Nuevas características • Añada soporte para la inferencia de clasificaciones de imágenes de muestra mediante TensorFlow Lite.
TensorFlow Detección de objetos Lite	Está disponible la versión 2.1.0 del nuevo componente de detección de objetos TensorFlow Lite. Nuevas características • Agregue soporte para la inferencia de detección de objetos de muestra mediante TensorFlow Lite.
TensorFlo w Tienda de modelos de clasificación de imágenes Lite	Está disponible la versión 2.1.0 del nuevo componente TensorFlow Lite de tienda de modelos de clasificación de imágenes. Nuevas características • Proporcione un modelo cuantificado MobileNet v1 previamente entrenado para inferir la clasificación de imágenes de muestra utilizando Lite. TensorFlow
TensorFlo w Tienda de modelos de detección de objetos Lite	Está disponible la versión 2.1.0 del nuevo componente TensorFlow Lite de tienda de modelos de detección de objetos. Nuevas características • Proporcione un MobileNet modelo de detección de disparo único (SSD) previamente entrenado en el conjunto de datos COCO para inferir la detección de objetos de muestra mediante Lite. TensorFlow
TensorFlow Lite	Está disponible la versión 2.5.0 del nuevo componente TensorFlow Lite. Nuevas características • Instale TensorFlow Lite v1.6.0 y sus dependencias en un entorno virtual en las plataformas Armv7, Armv8 () y x86_64. AArch64

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.0.5 el 9 de marzo de 2021

Esta versión proporciona la versión 2.0.5 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados. Soluciona un problema con la compatibilidad con el proxy de red y un problema con el punto final del plano de datos de Greengrass en las regiones de AWS China.

Fecha de lanzamiento: 09 de marzo de 2021

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	Está disponible la versión 2.0.5 del núcleo de Greengrass. Mejoras y correcciones de errores • Enruta correctamente el tráfico a través de un proxy de red configurado al descargar los componentes AWS proporcionados.

Componente	Detalles
	• Utilice el punto de conexión correcto del plano de datos de Greengrass en las regiones de AWS de China.

Lanzamiento: actualización del software AWS IoT Greengrass Core v2.0.4 el 4 de febrero de 2021

Este lanzamiento proporciona la versión 2.0.4 del componente de núcleo de Greengrass. Incluye el nuevo parámetro `greengrassDataPlanePort` para configurar la comunicación HTTPS a través del puerto 443 y corrige errores. La política de IAM mínima ahora requiere que se ejecute el instalador de software AWS IoT Greengrass principal `iam:GetPolicy` y `sts:GetCallerIdentity` cuando se ejecute con él. `--provision true`

Fecha de lanzamiento: 04 de febrero de 2021

Actualizaciones de componentes públicos

En la siguiente tabla se enumeran los componentes AWS proporcionados que incluyen funciones nuevas y actualizadas.

> **Important**
>
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
>
> Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Componente	Detalles
Núcleo de Greengrass	La versión 2.0.4 del núcleo de Greengrass está disponible. Nuevas características • Activa el tráfico HTTPS a través del puerto 443. Puede usar el nuevo parámetro de configuración `greengrassDataPlanePort` de la versión 2.0.4 del componente de núcleo para configurar la comunicación HTTPS para que viaje por el puerto 443 en lugar del puerto predeterm inado 8443. Para obtener más información, consulte Configuración de HTTPS a través del puerto 443. • Agrega la variable de receta de la ruta de trabajo. Puede utilizar esta variable de receta para obtener la ruta a las carpetas de trabajo de los componentes, que puede utilizar para compartir archivos entre los componentes y sus dependencias. Para obtener más información, consulte la variable de receta de ruta de trabajo. Mejoras y correcciones de errores • Impide la creación de la política de roles de intercambio de tokens AWS Identity and Access Management (IAM) si ya existe una política de roles. Como resultado de este cambio, el instalador ahora necesita `iam:GetPolicy` y `sts:GetCallerIdentity` cuando se ejecuta con `--provision true` . Para obtener más información, consulte Política de IAM mínima para que el instalador aprovisione recursos. • Gestiona correctamente la cancelación de una implementación que aún no se registró correctamente. • Actualiza la configuración para eliminar las entradas antiguas con marcas temporales más recientes al anular una implementación. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.

Migrar desde AWS IoT Greengrass la versión 1

AWS IoT Greengrass Version 2 es una versión principal del software AWS IoT Greengrass principal y APIs de la consola. AWS IoT Greengrass V2 introduce varias mejoras AWS IoT Greengrass V1, como las aplicaciones modulares, las implementaciones en grandes flotas de dispositivos y la compatibilidad con plataformas adicionales.

> **Note**
>
> Aviso de fin del soporte: el 7 de octubre de 2026, AWS finalizará el soporte para. AWS IoT Greengrass Version 1 Después del 7 de octubre de 2026, ya no podrás acceder a la AWS IoT Greengrass V1 consola ni a AWS IoT Greengrass V1 los recursos.

Siga las instrucciones de esta guía para migrar de AWS IoT Greengrass V1 a AWS IoT Greengrass V2.

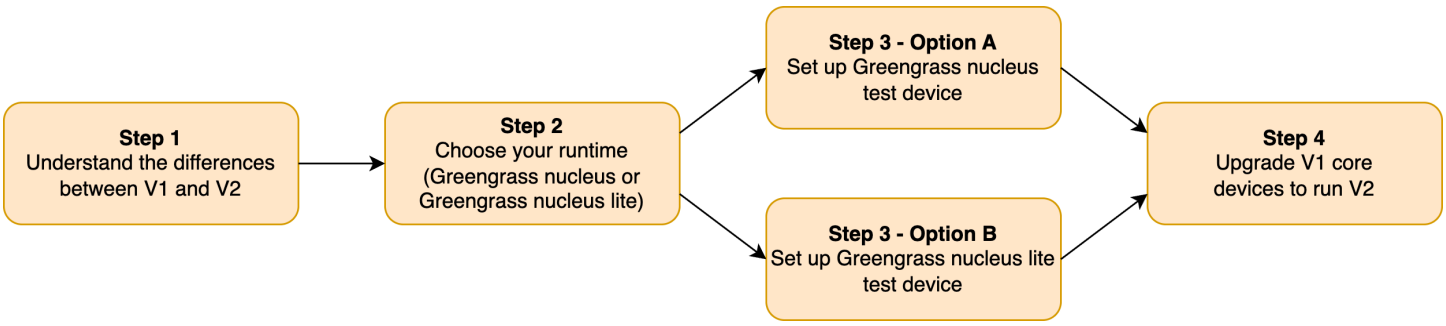
Información general sobre la migración

En un nivel superior, puede utilizar el siguiente procedimiento para actualizar los dispositivos principales de AWS IoT Greengrass V1 a AWS IoT Greengrass V2.

Antes de realizar la migración, deberá elegir entre dos opciones de tiempo de ejecución:

- Greengrass Nucleus (menor esfuerzo de migración, compatibilidad con todas las funciones)
- Greengrass nucleus lite (mayor esfuerzo de migración, diseñado para dispositivos con recursos limitados).

El procedimiento exacto que siga depende de los recursos del dispositivo, de las funciones requeridas y de los requisitos específicos del entorno.



1. Explicación de las diferencias entre V1 y V2

   AWS IoT Greengrass V2 presenta nuevos conceptos fundamentales para las flotas de dispositivos y el software desplegable, y la V2 simplifica varios conceptos de la V1.

   El servicio AWS IoT Greengrass V2 en la nube y el software AWS IoT Greengrass Core v2.x no son retrocompatibles con el servicio en la AWS IoT Greengrass V1 nube y el software Core v1.x. AWS IoT Greengrass Como resultado, las actualizaciones AWS IoT Greengrass V1 over-the-air (OTA) no pueden actualizar los dispositivos principales de la V1 a la V2.

2. Elige tu tiempo de ejecución (Greengrass nucleus o Greengrass nucleus lite)

   Decida entre Greengrass nucleus o Greengrass nucleus lite en función de los recursos y requisitos de funciones de su dispositivo:

   - Ruta del núcleo de Greengrass: menor esfuerzo migratorio. Las funciones de Lambda se pueden importar como componentes de Lambda con cambios de código mínimos. Admite las funciones de la versión 1 (servicio paralelo local, dispositivos cliente, conectores).
   - Ruta Greengrass Nucleus Lite: mayor esfuerzo de migración. Las funciones Lambda deben convertirse en componentes genéricos, lo que requiere cambios en el código para usar SDK para dispositivos con AWS IoT V2/AWS IoT Greengrass Component SDK en lugar de Core SDK. AWS IoT Greengrass No es compatible con el servicio paralelo local, los dispositivos cliente ni los conectores.

3. Configuración de un nuevo dispositivo para probar las aplicaciones de V1 en V2

   Para minimizar el riesgo de sus dispositivos en producción, cree un nuevo dispositivo para probar las aplicaciones de V1 en V2. Elija la guía de configuración en función de su selección de tiempo de ejecución:

   - Opción A: tiempo de ejecución de Greengrass Nucleus: configure un nuevo dispositivo para probar las aplicaciones de la V1 en la V2. Importe funciones de Lambda como componentes de Lambda con cambios de código mínimos.
   - Opción B: tiempo de ejecución de Greengrass nucleus lite: configure un nuevo dispositivo para probar las aplicaciones de la V1 en la V2 (Greengrass nucleus lite). Convierta funciones Lambda en componentes genéricos mediante. SDK para dispositivos con AWS IoT

4. Actualización de los dispositivos principales de V1 para ejecutar V2

   Tras realizar las pruebas en un dispositivo nuevo, actualice sus dispositivos principales V1 existentes para que ejecuten el software AWS IoT Greengrass Core v2.x y sus componentes.

AWS IoT Greengrass V2 Para migrar una flota de dispositivos de V1 a V2, repita este paso para cada dispositivo de la flota.

Diferencias entre los hosts de tipo AWS IoT Greengrass V1 y las AWS IoT Greengrass V2

AWS IoT Greengrass V2 presenta nuevos conceptos fundamentales para los dispositivos, las flotas y el software que se implementa. En esta sección, se describen los conceptos de la V1 que son diferentes en la V2.

Conceptos y terminología de Greengrass

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Código de la aplicación	En AWS IoT Greengrass V1, las funciones de Lambda definen el software que se ejecuta en los dispositivos principales. En cada grupo de Greengrass, usted define las suscripciones y los recursos locales que utiliza la función. Para las funciones de Lambda que el software AWS IoT Greengrass Core ejecuta en un entorno de tiempo de ejecución de Lambda en contenedores, debe definir los parámetros del contenedor, como los límites de memoria.	En AWS IoT Greengrass V2, los componentes son los módulos de software que se ejecutan en los dispositivos principales. • Cada componente tiene una receta que define los metadatos, parámetros, dependencias y scripts del componente que se ejecutarán en cada paso del ciclo de vida del component e. • La receta también define los artefactos del componente, que son archivos binarios, como scripts, código compilado y recursos estáticos. • Al implementar un componente en un dispositi vo principal, el dispositi

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		vo principal descarga la receta del componente y los artefactos para ejecutar el componente. Puede importar las funciones de Lambda de la versión 1 como componentes que se ejecutan en un entorno de tiempo de ejecución de Lambda en AWS IoT Greengrass V2. Al importar la función de Lambda, se especifican las suscripciones, los recursos locales y los parámetros del contenedor de la función. Para obtener más información, consulte Paso 2: Crear e implementar AWS IoT Greengrass V2 componentes para migrar aplicaciones AWS IoT Greengrass V1. Para obtener más información sobre cómo crear component es personalizados, consulte Desarrollo de componentes de AWS IoT Greengrass.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Grupos e implementaciones de AWS IoT Greengrass	En AWS IoT Greengrass V1, un grupo define el dispositivo principal, la configuración y el software de ese dispositivo principal y la lista de objetos de AWS IoT que se pueden conectar a ese dispositivo principal. Se crea una implementación para enviar la configuración de un grupo a un dispositivo principal.	En AWS IoT Greengrass V2, se utilizan las implementaciones para definir los componentes y las configuraciones de software que se ejecutan en los dispositivos principales. • Cada implementación se dirige a un dispositivo de un solo núcleo (que es un objeto de AWS IoT) o a un grupo de objetos de AWS IoT que puede contener varios dispositivos principales. • Las implementaciones en grupos de objetos son continuas, por lo que cuando se agrega un dispositivo principal a un grupo de objetos, recibe la configuración de software de ese grupo. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos. En AWS IoT Greengrass V2, también puede crear implementaciones locales mediante la CLI de Greengrass para probar componentes

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		de software personalizados en el dispositivo en el que los desarrolla. Para obtener más información, consulte Creación de componentes de AWS IoT Greengrass.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Software AWS IoT Greengrass Core	En AWS IoT Greengrass V1, el software AWS IoT Greengrass Core es un paquete único que contiene el software y todas sus características. El dispositivo de periferia en el que instala el software AWS IoT Greengrass Core se llama Greengrass principal.	En AWS IoT Greengrass V2, el software AWS IoT Greengrass Core es modular, por lo que puede elegir qué instalar para controlar el consumo de memoria. • El componente de núcleo de Greengrass es la instalación mínima requerida del software AWS IoT Greengrass Core. El dispositivo de periferia en el que instala el núcleo se llama dispositivo principal de Greengrass. • El núcleo se encarga de las implementaciones, la orquestación y la administración del ciclo de vida de otros componentes del dispositivo principal. • Las características como el administrador de flujos, el administrador de secretos y el administrador de registros son componentes que se implementan solo cuando se necesitan esas características. Para obtener más información, consulte Componentes proporcionados por AWS.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Connectors	En AWS IoT Greengrass V1, los conectores son módulos preconstruidos que implement a los dispositivos principal es de AWS IoT Greengras s V1 para interactuar con la infraestructura local, los protocolos de los dispositivos, AWS, y otros servicios en la nube.	En AWS IoT Greengrass V2, AWS brinda componentes de Greengrass que implementan la funcionalidad proporcionada por los conectores en la V1. Los siguientes componentes de AWS IoT Greengrass V2 ofrecen la funcionalidad del conector de Greengrass V1: • Componente de métricas de CloudWatch • Componente de AWS IoT Device Defender • Componente de Firehose • Conector adaptador de protocolo Modbus-RTU • Componente de Amazon SNS Para obtener más informaci ón, consulte Componentes proporcionados por AWS.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Dispositivos conectados (dispositivos de Greengrass)	En AWS IoT Greengrass V1, los dispositivos conectados son elementos de AWS IoT que se agregan a un grupo de Greengrass para conectarse al dispositivo principal de ese grupo y comunicarse a través de MQTT. Debe implement ar ese grupo cada vez que agregue o elimine un dispositi vo conectado. Utiliza las suscripciones para retransmi tir mensajes entre los dispositi vos conectados, AWS IoT Core y las aplicaciones del dispositivo principal.	En AWS IoT Greengrass V2, los dispositivos conectados se llaman dispositivos de cliente de Greengrass. • Los dispositivos de cliente se asocian a los dispositivos principales para conectarlos y comunicarse a través de MQTT. • Para autorizar la conexión de los dispositivos de cliente, debe definir políticas de autorización que se puedan aplicar a grupos de dispositivos de cliente, por lo que no es necesario crear una implementación para agregar o eliminar un dispositivo de cliente. • Para retransmitir mensajes entre los dispositivos de cliente, AWS IoT Core y los componentes de Greengras s, puede configurar un componente de puente MQTT opcional. En ambos AWS IoT Greengrass V1 y AWS IoT Greengrass V2, los dispositivos pueden ejecutar FreeRTOS o usar el SDK para dispositivos con AWS

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		IoT o la API de detección de Greengrass para obtener información sobre los dispositivos principales a los que se pueden conectar. La API de detección de Greengrass es compatible con versiones anteriores, por lo que si tiene dispositivos de cliente que se conectan a un dispositivo principal V1, puede conectarlos a un dispositivo principal V2 sin cambiar su código. Para obtener más información acerca de los dispositivos de cliente, consulte Interacción con dispositivos IoT locales.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Recursos locales	En AWS IoT Greengrass V1, las funciones de Lambda que se ejecutan en contenedores se pueden configurar para acceder a los volúmenes y dispositivos del sistema de archivos del dispositivo principal. Estos recursos del sistema de archivos se conocen como recursos locales.	En AWS IoT Greengrass V2, puede ejecutar componentes que sean funciones de Lambda, contenedores de Docker o procesos nativos del sistema operativo o tiempos de ejecución personalizados. • Al importar una función de Lambda conteneri zada como component e, debe especificar los recursos locales que utiliza la función. • Las funciones de Lambda no contenerizadas y los componentes que no son de Lambda pueden funcionar directamente con los recursos locales en los dispositivos principales, por lo que no es necesario especificar los recursos locales que utiliza el componente.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Servicio de sombra local	En AWS IoT Greengrass V1, el servicio de sombra local está activado de forma predeterminada y solo admite sombras clásicas sin nombre. El SDK de AWS IoT Greengrass Core se utiliza en las funciones de Lambda para interactuar con las sombras de los dispositivos.	En AWS IoT Greengrass V2, habilita el servicio de sombra local implementando el componente de administrador de sombras. • Puede utilizar el SDK para dispositivos con AWS IoT V2 en las funciones de Lambda y componentes personalizados para interactuar con las sombras de los dispositivos. • El servicio de sombra local admite sombras con nombre. • El servicio de sombra local permite eliminar las sombras y sincronizar las sombras eliminadas con AWS IoT Core. Para obtener más información, consulte Interacción con las sombras de dispositivo.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2

Suscripciones	En AWS IoT Greengrass V1, se definen las suscripciones para un grupo de Greengrass para especificar los canales de comunicación entre las funciones de Lambda, los conectores, los dispositivos conectados, el agente MQTT de AWS IoT Core y el servicio de sombra local. Las suscripciones especifican dónde reciben las funciones de Lambda los mensajes de eventos para utilizarlos como cargas útiles de funciones.	En AWS IoT Greengrass V2, se especifican los canales de comunicación sin utilizar suscripciones.
		• Los componentes administran sus propios canales de comunicación para interactuar con la mensajería de publicación y suscripción local, los mensajes MQTT de AWS IoT Core y el servicio de sombra local.
		  • Para desarrollar un componente que reaccione a los mensajes de otro componente o del agente MQTT de AWS IoT Core, puede utilizar las interfaces de comunicación entre procesos (IPC) para la mensajería de publicación y suscripción local y la mensajería MQTT de AWS IoT Core.
		  • Para desarrollar un componente que interactúe con el servicio de sombra local, puede utilizar la interfaz IPC para el servicio de sombra local.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		• En la configuración del componente, se definen políticas de autorización para especificar los temas y las sombras locales que el componente tiene permiso para usar. • Para configurar los canales de comunicación entre los dispositivos de cliente, el agente local de publicación/suscripción y el agente MQTT de AWS IoT Core, debe configurar e implementar el componente puente de MQTT. El componente puente de MQTT le permite interactuar con los dispositivos de cliente en los componentes y retransmitir mensajes entre los dispositivos de cliente y AWS IoT Core.

Concepto	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Acceso a otros Servicios de AWS	En AWS IoT Greengrass V1, se adjunta un rol de AWS Identity and Access Management (IAM), llamado rol de grupo, a un grupo de Greengrass. El rol de grupo define los permisos que utilizan las funciones de Lambda y características de AWS IoT Greengrass en el dispositivo principal de ese grupo para acceder a los Servicios de AWS.	En AWS IoT Greengrass V2, adjunta un alias de rol de AWS IoT a un dispositivo principal de Greengrass. El alias del rol apunta a un rol de IAM llamado rol de intercambio de token. El rol de intercambio de tokens define los permisos que utilizan los componentes de Greengrass del dispositivo principal para acceder a Servicios de AWS. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Elige tu tiempo de ejecución (Greengrass nucleus o Greengrass nucleus lite)

La elección entre el núcleo de Greengrass y el núcleo lite de Greengrass depende de los recursos del dispositivo y de las funciones que utilicen las funciones de Lambda. Revise la matriz de compatibilidad de fuentes de eventos de la siguiente tabla y, a continuación, utilice el diagrama de flujo de decisiones para determinar qué tiempo de ejecución es el adecuado para la migración. Para ver una comparación detallada de las características del núcleo de Greengrass y del núcleo lite de Greengrass, consulte Cómo elegir el tiempo de ejecución.

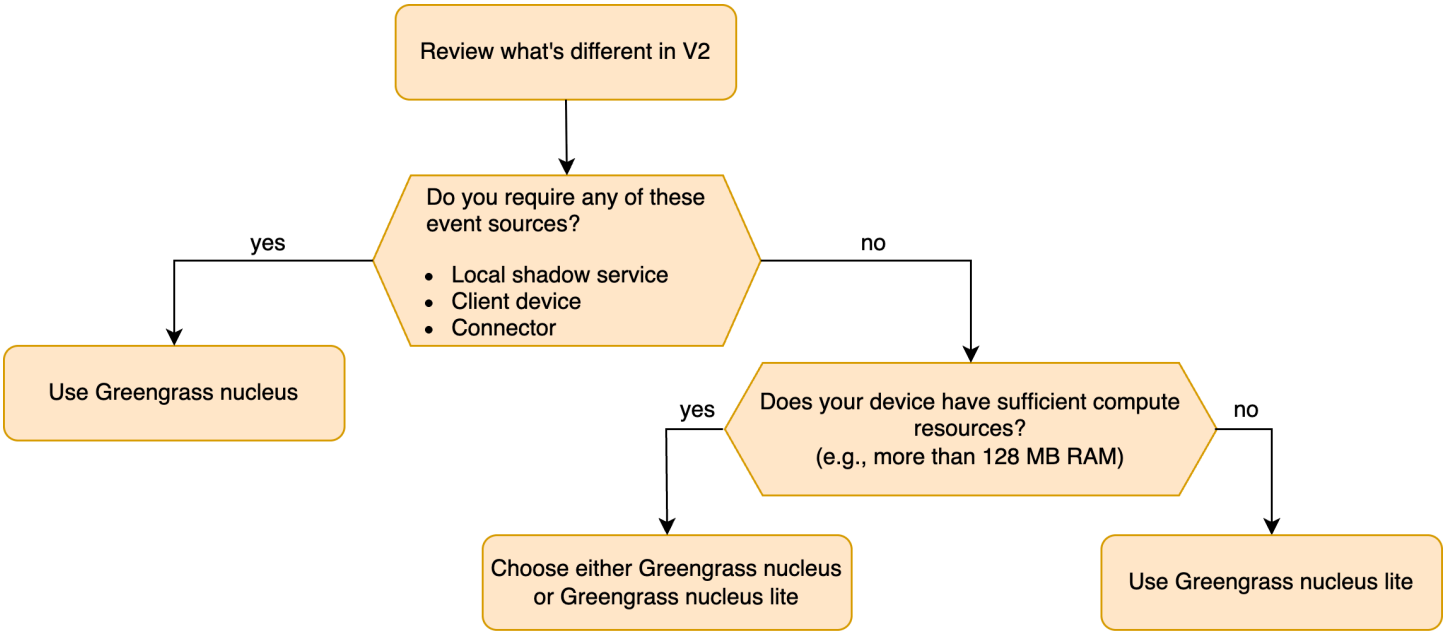
Matriz de compatibilidad de fuentes de eventos

En AWS IoT Greengrass V1, las funciones de Lambda se pueden comunicar con cinco tipos de fuentes de eventos: otras funciones de Lambda, un servicio paralelo local AWS IoT Core, dispositivos cliente y conectores. En la siguiente tabla se muestran cuáles de estas fuentes de eventos son compatibles en cada entorno de ejecución de la V2.

Nota: Los nombres de las fuentes de eventos utilizan AWS IoT Greengrass V1 terminología. Al migrar a la versión 2, las funciones de Lambda se convierten en componentes Lambda (compatibles únicamente con Greengrass nucleus) o en componentes genéricos (compatibles con Greengrass nucleus y Greengrass nucleus lite).

Origen del evento	Núcleo de Greengrass	Versión lite del núcleo de Greengrass
Otras funciones Lambda del grupo	✓ (componentes Lambda y componentes genéricos)	✓ (solo componentes genéricos)
AWS IoT Core servicio	✓	✓
Servicio de sombra local	✓	✗
Dispositivo cliente	✓	✗
Connector	✓	✗

flujo de decisiones de selección en tiempo de ejecución



Notas

1. Para conocer los requisitos de Greengrass nucleus lite y los detalles de compatibilidad, consulte Greengrass nucleus lite. Greengrass nucleus lite requiere un mínimo de 5 MB de RAM y está diseñado para dispositivos con recursos limitados.
2. El flujo de decisiones proporciona orientación basada en casos de uso típicos, pero no es un requisito estricto. Los clientes con dispositivos con recursos limitados y suficientes pueden optar por utilizar un único tiempo de ejecución en todos los dispositivos para simplificar el funcionamiento, incluso si algunos dispositivos pueden soportar cualquiera de los dos tiempos de ejecución.

Siguientes pasos

Tras elegir el tiempo de ejecución, proceda a configurar el dispositivo de prueba:

- Para el tiempo de ejecución de Greengrass Nucleus: configure un nuevo dispositivo para probar las aplicaciones de la V1 en la V2
- Para el tiempo de ejecución de Greengrass nucleus lite: configure un nuevo dispositivo con Greengrass nucleus lite

Configuración de un nuevo dispositivo para probar las aplicaciones de V1 en V2

Para minimizar el riesgo para sus dispositivos de producción, cree un nuevo dispositivo para probar las aplicaciones de la versión 1 en la versión 2 antes de actualizar los dispositivos de producción.

Elija una de las siguientes guías de configuración en función del tiempo de ejecución que haya seleccionado en el paso anterior:

- Opción A: Tiempo de ejecución del núcleo de Greengrass: siga esta opción Configuración de un nuevo dispositivo para probar las aplicaciones de V1 en V2 si ha elegido el núcleo de Greengrass. Con esta opción, puede importar funciones de Lambda como componentes de Lambda con cambios de código mínimos y admite funciones de la versión 1, como el servicio paralelo local, los dispositivos cliente y los conectores.
- Opción B: Tiempo de ejecución de Greengrass nucleus lite: siga esta opción Configurar un nuevo dispositivo para probar las aplicaciones de la V1 en la V2 (Greengrass nucleus lite) si ha elegido

Greengrass nucleus lite. Esta opción requiere convertir las funciones de Lambda en componentes genéricos mediante la SDK para dispositivos con AWS IoT V2 o el SDK de AWS IoT Greengrass componentes, pero está optimizada para dispositivos con recursos limitados.

Configuración de un nuevo dispositivo para probar las aplicaciones de V1 en V2

Configure un nuevo dispositivo AWS IoT Greengrass V2 principal para implementar y probar los componentes y AWS Lambda funciones AWS proporcionados para sus AWS IoT Greengrass V1 aplicaciones. También puede usar este dispositivo principal V2 para desarrollar y probar componentes de Greengrass personalizados adicionales que ejecutan procesos nativos en los dispositivos principales. Después de probar sus aplicaciones en un dispositivo principal V2, puede actualizar sus dispositivos principales V1 existentes a V2 e implementar los componentes de V2 que ofrecen la funcionalidad de la V1.

Paso 1: Instalar AWS IoT Greengrass V2 en un dispositivo nuevo

Instale el software AWS IoT Greengrass Core v2.x en un dispositivo nuevo. Puede seguir el tutorial de introducción para configurar un dispositivo y aprender a desarrollar e implementar componentes. En este tutorial, se utiliza el aprovisionamiento automático para configurar rápidamente un dispositivo. Cuando instale el software AWS IoT Greengrass Core v2.x, especifique el `--deploy-dev-tools` argumento para implementar la CLI de Greengrass, de modo que pueda desarrollar, probar y depurar componentes directamente en el dispositivo. Para obtener más información sobre otras opciones de instalación, incluida la forma de instalar el software AWS IoT Greengrass Core detrás de un proxy o mediante un módulo de seguridad de hardware (HSM), consulte. Instalación del software AWS IoT Greengrass Core

(Opcional) Habilita el registro en Amazon CloudWatch Logs

Para permitir que un dispositivo principal V2 cargue registros en Amazon CloudWatch Logs, puede implementar el componente AWS de administrador de registros proporcionado. Puede utilizar CloudWatch los registros para ver los registros de los componentes, de forma que pueda depurar y solucionar problemas sin tener acceso al sistema de archivos del dispositivo principal. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Paso 2: Crear e implementar AWS IoT Greengrass V2 componentes para migrar aplicaciones AWS IoT Greengrass V1

Puede ejecutar la mayoría de AWS IoT Greengrass V1 las aplicaciones en ellas AWS IoT Greengrass V2. Puede importar funciones Lambda como componentes en AWS IoT Greengrass V2 los que se ejecutan y puede usar los componentes AWS proporcionados que ofrecen la misma funcionalidad que los conectores. AWS IoT Greengrass

También puede desarrollar componentes personalizados para crear cualquier característica o tiempo de ejecución que se ejecute en los dispositivos principales de Greengrass. Para obtener más información acerca de cómo desarrollar y probar componentes de forma local, consulte Creación de componentes de AWS IoT Greengrass.

Temas

- Cómo importar una función de Lambda V1
- Uso de conectores de V1
- Ejecución de contenedores de Docker
- Conexión de dispositivos de Greengrass V1
- Cómo habilitar el servicio de sombra local
- Intégrelo con AWS IoT SiteWise

Cómo importar una función de Lambda V1

Puede importar funciones Lambda como AWS IoT Greengrass V2 componentes. Puede elegir entre los siguientes enfoques:

- Importe funciones de Lambda V1 directamente como componentes de Greengrass.
- Actualice las funciones de Lambda para utilizar las bibliotecas de Greengrass en la SDK para dispositivos con AWS IoT versión 2 y, a continuación, importe las funciones de Lambda como componentes de Greengrass.
- Cree componentes personalizados que utilicen código que no sea de Lambda y la SDK para dispositivos con AWS IoT versión 2 para implementar la misma funcionalidad que sus funciones de Lambda.

Si su función Lambda usa funciones, como el administrador de transmisiones o secretos locales, debe definir las dependencias en los componentes AWS proporcionados que empaquetan estas

funciones. Al implementar el componente de función de Lambda, la implementación también incluye el componente para cada característica que defina como dependencia. En la implementación, puede configurar los parámetros, como los secretos que se van a implementar en el dispositivo principal. No todas las características de la V1 requieren una dependencia de componentes para la función de Lambda en la V2. La siguiente lista describe cómo utilizar las funciones de la V1 en el componente de la función Lambda de la V2:

- Acceda a otros servicios AWS

  Si la función Lambda usa AWS credenciales para realizar solicitudes a otros AWS servicios, la función de intercambio de token del dispositivo principal debe permitir que el dispositivo principal realice las AWS operaciones que usa la función Lambda. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

- Administrador de flujos

  Si su función de Lambda usa el administrador de flujos, especifique `aws.greengrass.StreamManager` como una dependencia de componente al importar la función. Al implementar el componente del administrador de flujos, especifique los parámetros del administrador de flujos que desee configurar para los dispositivos principales de destino. La función de intercambio de tokens del dispositivo principal debe permitir que el dispositivo principal acceda a los Nube de AWS destinos que utilices con Stream Manager. Para obtener más información, consulte Administrador de flujos.

- Secretos locales

  Si su función de Lambda usa el administrador de flujos, especifíque `aws.greengrass.SecretManager` como una dependencia de componente al importar la función. Al implementar el componente del administrador de secretos, especifique los recursos secretos que se van a implementar en los dispositivos principales de destino. El rol de intercambio de tokens del dispositivo principal debe permitir que el dispositivo principal recupere los recursos secretos para implementarlos. Para obtener más información, consulte Administrador de secretos.

  Al implementar el componente de la función Lambda, configúrelo para que tenga una política de autorización de IPC que conceda permiso para usar la operación de GetSecretValue IPC en la V2. SDK para dispositivos con AWS IoT

- Sombras locales

  Si la función Lambda interactúa con las sombras locales, debe actualizar el código de la función Lambda para usar la V2. SDK para dispositivos con AWS IoT También debe especificar

`aws.greengrass.ShadowManager` como dependencia de un componente al importar la función. Para obtener más información, consulte Interacción con las sombras de dispositivo.

Al implementar el componente de la función Lambda, configúrelo para que tenga una política de autorización de IPC que conceda permiso para utilizar las operaciones de IPC ocultas en la V2. SDK para dispositivos con AWS IoT

- Suscripciones
  - Si su función de Lambda se suscribe a mensajes de un origen en la nube, especifique esas suscripciones como orígenes de eventos al importar la función.
  - Si la función de Lambda se suscribe a los mensajes de otra función de Lambda, o si la función de Lambda publica mensajes en u AWS IoT Core otras funciones de Lambda, configure e implemente el componente del router de suscripciones heredado al implementar la función de Lambda. Al implementar el componente del enrutador de suscripciones heredado, especifique las suscripciones que utiliza la función de Lambda.

    > **Note**
    >
    > El componente de router de suscripción antiguo solo es necesario si la función Lambda utiliza la `publish()` función del SDK AWS IoT Greengrass principal. Si actualiza el código de la función Lambda para utilizar la interfaz de comunicación entre procesos (IPC) de la SDK para dispositivos con AWS IoT V2, no necesitará implementar el componente de router de suscripción heredado. Para obtener más información, consulte los siguientes servicios de comunicación entre procesos:
    >
    > - Publicar/suscribir mensajes locales
    > - Publicar/suscribir mensajes MQTT AWS IoT Core

  - Si su función de Lambda se suscribe a mensajes de dispositivos conectados locales, especifique esas suscripciones como orígenes de eventos al importar la función. También debe configurar e implementar el componente de puente MQTT para retransmitir los mensajes desde los dispositivos conectados a los publish/subscribe temas locales que especifique como fuentes de eventos.
  - Si su función Lambda publica mensajes en dispositivos conectados localmente, debe actualizar el código de la función Lambda para usar la SDK para dispositivos con AWS IoT V2 para publicar mensajes locales. publish/subscribe También debe configurar e implementar el componente de puente MQTT para retransmitir los mensajes desde el intermediario de publish/subscribe mensajes local a los dispositivos conectados.

- Volúmenes y dispositivos locales

  Si la función de Lambda en contenedores accede a volúmenes o dispositivos locales, especifique esos volúmenes y dispositivos al importar la función de Lambda. Esta característica no requiere una dependencia de componentes.

Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

Uso de conectores de V1

Puede implementar los componentes AWS proporcionados que ofrecen la misma funcionalidad que algunos AWS IoT Greengrass conectores. Al crear la implementación, puede configurar los parámetros de los conectores.

Los siguientes AWS IoT Greengrass V2 componentes proporcionan la funcionalidad del conector Greengrass V1:

- CloudWatch componente de métricas
- AWS IoT Device Defender component
- Componente de Firehose
- Conector adaptador de protocolo Modbus-RTU
- Componente de Amazon SNS

Ejecución de contenedores de Docker

AWS IoT Greengrass V2 no proporciona un componente que sustituya directamente al conector de implementación de aplicaciones Docker de la versión 1. Sin embargo, puede usar el componente del administrador de aplicaciones de Docker para descargar imágenes de Docker y, a continuación, crear componentes personalizados que ejecuten contenedores de Docker a partir de las imágenes descargadas. Para obtener más información, consulte Ejecución de un contenedor de Docker y Administrador de aplicaciones de Docker.

Conexión de dispositivos de Greengrass V1

Los dispositivos conectados en AWS IoT Greengrass V1 se denominan dispositivos cliente en AWS IoT Greengrass V2. AWS IoT Greengrass V2 la compatibilidad con dispositivos cliente es compatible con versiones anteriores AWS IoT Greengrass V1, por lo que puede conectar los dispositivos

cliente V1 a los dispositivos principales V2 sin cambiar su código de aplicación. Para permitir que los dispositivos de cliente se conecten a un dispositivo principal V2, implemente componentes de Greengrass que admiten dispositivos de cliente y asocie los dispositivos de cliente al dispositivo principal. Para retransmitir mensajes (incluidas las funciones de Lambda) entre los dispositivos cliente, el servicio en la nube de AWS IoT Core y los componentes de Greengrass, implemente y configure el componente puente MQTT. Implemente el componente detector de IP para detectar de forma automática la información de conectividad o administre manualmente los puntos de conexión. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Cómo habilitar el servicio de sombra local

En AWS IoT Greengrass V2, el servicio de sombra local se implementa mediante el componente de administrador AWS de sombras proporcionado. AWS IoT Greengrass V2 también incluye soporte para sombras con nombre. Para permitir que sus componentes interactúen con las sombras locales y sincronicen los estados de las sombras AWS IoT Core, configure e implemente el componente administrador de sombras y utilice las operaciones de IPC ocultas en el código del componente. Para obtener más información, consulte Interacción con las sombras de dispositivo.

Intégrelo con AWS IoT SiteWise

Si utilizas tu dispositivo principal V1 como AWS IoT SiteWise puerta de enlace, sigue las instrucciones para configurar tu nuevo dispositivo central V2 como AWS IoT SiteWise puerta de enlace. AWS IoT SiteWise proporciona un script de instalación que despliega los AWS IoT SiteWise componentes automáticamente.

Paso 3: Pruebe sus aplicaciones AWS IoT Greengrass V2

Tras crear e implementar los componentes de la V2 en el nuevo dispositivo principal de la V2, compruebe que las aplicaciones cumplen sus expectativas. Puede consultar los registros del dispositivo para ver los mensajes de salida estándar (stdout) y de error estándar (stderr) de sus componentes. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Si implementó la CLI de Greengrass en el dispositivo principal, puede usarla para depurar componentes y sus configuraciones. Para obtener más información, consulte Comandos de la CLI de Greengrass.

Tras comprobar que las aplicaciones funcionan en un dispositivo principal V2, puede implementar los componentes de Greengrass de la aplicación en otros dispositivos principales. Si ha desarrollado

componentes personalizados que ejecutan procesos nativos o contenedores de Docker, primero debe publicar esos componentes en el AWS IoT Greengrass servicio para implementarlos en otros dispositivos principales.

Configurar un nuevo dispositivo para probar las aplicaciones de la V1 en la V2 (Greengrass nucleus lite)

Configure un nuevo dispositivo con el núcleo lite de Greengrass para probar los componentes genéricos que cree para migrar sus AWS IoT Greengrass V1 aplicaciones a la versión 2. Greengrass nucleus lite es un tiempo de ejecución ligero optimizado para dispositivos con recursos limitados. Puede usar este dispositivo para desarrollar y probar componentes personalizados de Greengrass que ejecutan procesos nativos. Después de probar sus aplicaciones en un dispositivo Greengrass nucleus lite, puede implementar los componentes V2 en otros dispositivos que ejecuten Greengrass nucleus lite o actualizar sus dispositivos centrales V1 existentes a V2.

Paso 1: Instalar Greengrass nucleus lite en un dispositivo nuevo

Instale el núcleo lite de Greengrass en un dispositivo nuevo. Siga la guía de instalación de Greengrass nucleus lite para configurar un dispositivo.

> **Note**
>
> Greengrass nucleus lite ya no es compatible con el servicio paralelo local, los dispositivos cliente ni los conectores. Asegúrese de que sus aplicaciones de la versión 1 no dependan de estas funciones antes de continuar con esta guía.

Paso 2: Crear e implementar componentes genéricos para migrar las funciones de AWS IoT Greengrass V1 Lambda

Para replicar la funcionalidad de sus funciones AWS IoT Greengrass V1 Lambda en Greengrass nucleus lite, debe convertirlas en componentes genéricos. Esto implica reescribir el código de la función Lambda para usar SDK para dispositivos con AWS IoT la V2 AWS IoT Greengrass o el SDK de componentes en lugar AWS IoT Greengrass del SDK principal.

En la siguiente tabla se enumeran los ejemplos de componentes SDKs utilizados en la V2 de esta guía:

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0
AWS IoT Greengrass SDK de componentes (C/C++) (actualmente en versión preliminar)	v0.4.0

Los ejemplos siguientes muestran las funciones de Lambda que utilizan el SDK AWS IoT Greengrass V1 principal y sus componentes genéricos equivalentes, con código de componente, recetas e instrucciones de compilación en varios lenguajes de programación para dos escenarios principales:

- Comunicación local: componentes que se comunican con otros componentes del mismo dispositivo mediante un pub/sub local
- Comunicación en la nube: componentes que se comunican con AWS IoT Core u otros servicios AWS

Escenario 1: Comunicación local (editor → procesador → suscriptor)

En este escenario se muestra la conversión de una función Lambda V1 que utiliza pub/sub comunicación local en un componente genérico V2.

Arquitectura de aplicaciones

Este ejemplo incluye tres componentes:

- La editorial Lambda publica los datos de temperatura
- El procesador Lambda recibe los datos y los procesa
- El procesador Lambda publica el resultado procesado al suscriptor Lambda

El siguiente ejemplo de código se centra en el procesador Lambda, que muestra tanto la suscripción como la publicación de mensajes para la comunicación local.

Suscripciones grupales V1

En AWS IoT Greengrass V1, las siguientes suscripciones grupales permiten la comunicación entre las funciones de Lambda:

Suscripción 1: editor → procesador

- Fuente: Lambda (editorial)
- Objetivo: Lambda (procesador)
- Tema: sensores/temperatura

Suscripción 2: procesador → suscriptor

- Fuente: Lambda (procesador)
- Destino: Lambda (suscriptor)
- Tema: lambda/alerts

Función Lambda del procesador (V1)

Python

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')

def lambda_handler(event, context):
    """
    Receives temperature from publisher Lambda,
    processes it, and forwards to subscriber Lambda
    """
    # Receive from publisher Lambda.
    sensor_id = event['sensor_id']
    temperature = event['temperature']

    print(f"Received from sensor {sensor_id}: {temperature}°F")
```

```
    # Process: Check if temperature is high.
    if temperature > 80:
        alert_data = {
            'sensor_id': sensor_id,
            'temperature': temperature,
            'alert': 'HIGH_TEMPERATURE'
        }

        # Publish to another Lambda using greengrasssdk.
        iot_client.publish(
            topic='lambda/alerts',
            payload=json.dumps(alert_data)
        )

        print(f"Alert sent to subscriber Lambda")

    return {'statusCode': 200}
```

Java

```
import com.amazonaws.greengrass.javasdk.IotDataClient;
import com.amazonaws.greengrass.javasdk.model.PublishRequest;
import com.amazonaws.services.lambda.runtime.Context;
import com.google.gson.Gson;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;

public class TemperatureProcessorLambda {
    private static final Gson gson = new Gson();
    private final IotDataClient iotDataClient;

    public TemperatureProcessorLambda() {
        this.iotDataClient = new IotDataClient();
    }

    public String handleRequest(Map<String, Object> event, Context context) {
        /*
         * Receives temperature from publisher Lambda,
         * processes it, and forwards to subscriber Lambda
         */
```

```
        // Receive from publisher Lambda.
        String sensorId = (String) event.get("sensor_id");
        Number temp = (Number) event.get("temperature");
        int temperature = temp.intValue();

        System.out.println("Received from sensor " + sensorId + ": " + temperature +
 "°F");

        // Process: Check if temperature is high.
        if (temperature > 80) {
            Map<String, Object> alertData = new HashMap<>();
            alertData.put("sensor_id", sensorId);
            alertData.put("temperature", temperature);
            alertData.put("alert", "HIGH_TEMPERATURE");

            // Publish to another Lambda using greengrasssdk.
            String payload = gson.toJson(alertData);
            PublishRequest publishRequest = new PublishRequest()
                .withTopic("lambda/alerts")

 .withPayload(ByteBuffer.wrap(payload.getBytes(StandardCharsets.UTF_8)));

            iotDataClient.publish(publishRequest);

            System.out.println("Alert sent to subscriber Lambda");
        }

        return "Success";
    }
}
```

JavaScript

```
const greengrasssdk = require('aws-greengrass-core-sdk');

const iotClient = new greengrasssdk.IotData();

/**
 * Greengrass v1 Lambda function
 * Receives temperature from publisher Lambda,
 * processes it, and forwards to subscriber Lambda
 */
exports.handler = function(event, context) {
```

```
    // Receive from publisher Lambda.
    const sensorId = event.sensor_id;
    const temperature = event.temperature;

    console.log(`Received from sensor ${sensorId}: ${temperature}°F`);

    // Process: Check if temperature is high.
    if (temperature > 80) {
        const alertData = {
            sensor_id: sensorId,
            temperature: temperature,
            alert: 'HIGH_TEMPERATURE'
        };

        // Publish to another Lambda using greengrasssdk.
        const params = {
            topic: 'lambda/alerts',
            payload: JSON.stringify(alertData)
        };

        iotClient.publish(params, (err) => {
            if (err) {
                console.error('Error publishing alert:', err);
                context.fail(err);
            } else {
                console.log('Alert sent to subscriber Lambda');
                context.succeed('Success');
            }
        });
    } else {
        context.succeed('Success');
    }
};
```

C

```
#include <aws/greengrass/greengrasssdk.h>
#include <stdio.h>
#include <string.h>
#include <jansson.h>  // For JSON parsing.

static aws_greengrass_iot_data_client *iot_client = NULL;
```

```
void on_message_received(const char *topic, const uint8_t *payload, size_t
 payload_len, void *user_data) {
    // Parse the incoming message.
    char *payload_str = strndup((char *)payload, payload_len);
    json_error_t error;
    json_t *event = json_loads(payload_str, 0, &error);
    free(payload_str);

    if (!event) {
        fprintf(stderr, "Error parsing JSON: %s\n", error.text);
        return;
    }

    // Receive from publisher Lambda.
    json_t *sensor_id_obj = json_object_get(event, "sensor_id");
    json_t *temperature_obj = json_object_get(event, "temperature");

    const char *sensor_id = json_string_value(sensor_id_obj);
    int temperature = json_integer_value(temperature_obj);

    printf("Received from sensor %s: %d°F\n", sensor_id, temperature);

    // Process: Check if temperature is high.
    if (temperature > 80) {
        // Create alert data.
        json_t *alert_data = json_object();
        json_object_set_new(alert_data, "sensor_id", json_string(sensor_id));
        json_object_set_new(alert_data, "temperature", json_integer(temperature));
        json_object_set_new(alert_data, "alert", json_string("HIGH_TEMPERATURE"));

        // Convert to JSON string.
        char *alert_payload = json_dumps(alert_data, JSON_COMPACT);

        // Publish to another Lambda using greengrasssdk.
        aws_greengrass_publish_params params = {
            .topic = "lambda/alerts",
            .payload = (uint8_t *)alert_payload,
            .payload_len = strlen(alert_payload)
        };

        aws_greengrass_iot_data_publish(iot_client, &params);

        printf("Alert sent to subscriber Lambda\n");
```

```
        free(alert_payload);
        json_decref(alert_data);
    }

    json_decref(event);
}

int main(int argc, char *argv[]) {
    // Initialize Greengrass SDK.
    iot_client = aws_greengrass_iot_data_client_new();

    // Subscribe to temperature sensor topic.
    aws_greengrass_subscribe_params subscribe_params = {
        .topic = "sensors/temperature",
        .callback = on_message_received,
        .user_data = NULL
    };

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

    printf("Temperature Processor Lambda started\n");
    printf("Subscribed to sensors/temperature\n");
    printf("Waiting for sensor data...\n");

    // Keep the Lambda running.
    while (1) {
        sleep(1);
    }

    return 0;
}
```

C++

```
#include <aws/greengrass/greengrasssdk.h>
#include <iostream>
#include <string>
#include <memory>
#include <jansson.h> // For JSON parsing.
#include <unistd.h>

class TemperatureProcessor {
private:
```

```
    std::unique_ptr<aws_greengrass_iot_data_client,
                    decltype(&aws_greengrass_iot_data_client_destroy)> iot_client;

    static void message_callback_wrapper(const char *topic,
                                         const uint8_t *payload,
                                         size_t payload_len,
                                         void *user_data) {
        auto* processor = static_cast<TemperatureProcessor*>(user_data);
        processor->on_message_received(topic, payload, payload_len);
    }

public:
    TemperatureProcessor()
        : iot_client(aws_greengrass_iot_data_client_new(),
                     aws_greengrass_iot_data_client_destroy) {
        if (!iot_client) {
            throw std::runtime_error("Failed to create Greengrass IoT client");
        }
    }

    void on_message_received(const char *topic,
                             const uint8_t *payload,
                             size_t payload_len) {
        // Parse the incoming message.
        std::string payload_str(reinterpret_cast<const char*>(payload),
 payload_len);

        json_error_t error;
        json_t *event = json_loads(payload_str.c_str(), 0, &error);

        if (!event) {
            std::cerr << "Error parsing JSON: " << error.text << std::endl;
            return;
        }

        json_t *sensor_id_obj = json_object_get(event, "sensor_id");
        json_t *temperature_obj = json_object_get(event, "temperature");

        const char *sensor_id = json_string_value(sensor_id_obj);
        int temperature = json_integer_value(temperature_obj);

        std::cout << "Received from sensor " << sensor_id
                  << ": " << temperature << "ºF" << std::endl;
```

```
        if (temperature > 80) {
            send_alert(sensor_id, temperature);
        }

        json_decref(event);
    }

    void send_alert(const char *sensor_id, int temperature) {
        // Create alert data.
        json_t *alert_data = json_object();
        json_object_set_new(alert_data, "sensor_id", json_string(sensor_id));
        json_object_set_new(alert_data, "temperature", json_integer(temperature));
        json_object_set_new(alert_data, "alert", json_string("HIGH_TEMPERATURE"));

        // Convert to JSON string.
        char *alert_payload = json_dumps(alert_data, JSON_COMPACT);

        // Publish to another Lambda using greengrasssdk.
        aws_greengrass_publish_params params = {
            .topic = "lambda/alerts",
            .payload = reinterpret_cast<uint8_t*>(alert_payload),
            .payload_len = strlen(alert_payload)
        };

        aws_greengrass_iot_data_publish(iot_client.get(), &params);

        std::cout << "Alert sent to subscriber Lambda" << std::endl;

        free(alert_payload);
        json_decref(alert_data);
    }

    void subscribe_to_topic(const std::string& topic) {
        aws_greengrass_subscribe_params subscribe_params = {
            .topic = topic.c_str(),
            .callback = message_callback_wrapper,
            .user_data = this
        };

        aws_greengrass_iot_data_subscribe(iot_client.get(), &subscribe_params);

        std::cout << "Temperature Processor Lambda started" << std::endl;
        std::cout << "Subscribed to " << topic << std::endl;
        std::cout << "Waiting for sensor data..." << std::endl;
```

```
    }

    void run() {
        // Keep the Lambda running.
        while (true) {
            sleep(1);
        }
    }
};

int main(int argc, char *argv[]) {
    try {
        TemperatureProcessor processor;
        processor.subscribe_to_topic("sensors/temperature");
        processor.run();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Componente genérico (V2)

Para lograr la misma funcionalidad en AWS IoT Greengrass V2, cree un componente genérico con lo siguiente:

1. Código de componente

Python

Requisitos previos: Antes de usar el código de este componente, instale y verifique el código SDK para dispositivos con AWS IoT para Python en su dispositivo Greengrass:

```
# Install the SDK
pip3 install awsiotsdk

# Verify installation
python3 -c "import awsiot.greengrasscoreipc.clientv2; print('SDK installed
 successfully')"
```

Si encuentra conflictos de dependencia durante la instalación, intente instalar una versión específica del. SDK para dispositivos con AWS IoT

Si el comando de verificación muestra «El SDK se instaló correctamente», estás listo para usar el código del componente:

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    JsonMessage
)
import time

ipc_client = GreengrassCoreIPCClientV2()

def on_sensor_data(event):
    """
    Receives temperature from sensor publisher component,
    processes it, and forwards to alert component
    """
    try:
        # Receive from publisher component.
        data = event.json_message.message
        sensor_id = data['sensor_id']
        temperature = data['temperature']

        print(f"Received from sensor {sensor_id}: {temperature}°F")

        # Process: Check if temperature is high.
        if temperature > 80:
            alert_data = {
                'sensor_id': sensor_id,
                'temperature': temperature,
                'alert': 'HIGH_TEMPERATURE'
            }

            # Publish to another component (AlertHandler).
            ipc_client.publish_to_topic(
                topic='component/alerts',
                publish_message=PublishMessage(
                    json_message=JsonMessage(message=alert_data)
                )
            )
```

```
            print(f"Alert sent to AlertHandler component")

    except Exception as e:
        print(f"Error processing sensor data: {e}")

def main():
    print("Temperature Processor component starting...")

    # Subscribe to sensor data from publisher component.
    ipc_client.subscribe_to_topic(
        topic='sensors/temperature',
        on_stream_event=on_sensor_data
    )

    print("Subscribed to sensors/temperature")
    print("Waiting for sensor data...")

    # Keep running.
    while True:
        time.sleep(1)

if __name__ == '__main__':
    main()
```

Java

```
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.JsonMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import java.util.HashMap;
import java.util.Map;
import java.util.Optional;

public class TemperatureProcessor {
    private static GreengrassCoreIPCClientV2 ipcClient;

    public static void main(String[] args) {
        System.out.println("Temperature Processor component starting...");
```

```
        try (GreengrassCoreIPCClientV2 client =
GreengrassCoreIPCClientV2.builder().build()) {
            ipcClient = client;

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest()
                .withTopic("sensors/temperature");

            ipcClient.subscribeToTopic(
                subscribeRequest,
                TemperatureProcessor::onSensorData,
                Optional.empty(),
                Optional.empty()
            );

            System.out.println("Subscribed to sensors/temperature");
            System.out.println("Waiting for sensor data...");

            while (true) {
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    public static void onSensorData(SubscriptionResponseMessage message) {
        try {
            Map<String, Object> data = message.getJsonMessage().getMessage();
            String sensorId = (String) data.get("sensor_id");
            Number temp = (Number) data.get("temperature");
            int temperature = temp.intValue();

            System.out.println("Received from sensor " + sensorId + ": " +
temperature + "F");

            if (temperature > 80) {
                Map<String, Object> alertData = new HashMap<>();
                alertData.put("sensor_id", sensorId);
                alertData.put("temperature", temperature);
                alertData.put("alert", "HIGH_TEMPERATURE");

                JsonMessage jsonMessage = new JsonMessage().withMessage(alertData);
```

```
                PublishMessage publishMessage = new
 PublishMessage().withJsonMessage(jsonMessage);
                PublishToTopicRequest publishRequest = new PublishToTopicRequest()
                    .withTopic("component/alerts")
                    .withPublishMessage(publishMessage);

                ipcClient.publishToTopic(publishRequest);
                System.out.println("Alert sent to AlertHandler component");
            }
        } catch (Exception e) {
            System.err.println("Error processing sensor data: " + e.getMessage());
        }
    }
}
```

JavaScript

```
const greengrasscoreipc = require('aws-iot-device-sdk-v2').greengrasscoreipc;

class TemperatureProcessor {
    constructor() {
        this.ipcClient = null;
    }

    async start() {
        console.log('Temperature Processor component starting...');

        try {
            this.ipcClient = greengrasscoreipc.createClient();
            await this.ipcClient.connect();

            const subscribeRequest = {
                topic: 'sensors/temperature'
            };

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeRequest);

            streamingOperation.on('message', (message) => {
                this.onSensorData(message);
            });

            streamingOperation.on('streamError', (error) => {
```

```
            console.error('Stream error:', error);
        });

        streamingOperation.on('ended', () => {
            console.log('Subscription stream ended');
        });

        await streamingOperation.activate();

        console.log('Subscribed to sensors/temperature');
        console.log('Waiting for sensor data...');

    } catch (error) {
        console.error('Error starting component:', error);
        process.exit(1);
    }
}

async onSensorData(message) {
    try {
        const data = message.jsonMessage.message;

        const sensorId = data.sensor_id;
        const temperature = data.temperature;

        console.log(`Received from sensor ${sensorId}: ${temperature}°F`);

        if (temperature > 80) {
            const alertData = {
                sensor_id: sensorId,
                temperature: temperature,
                alert: 'HIGH_TEMPERATURE'
            };

            const publishRequest = {
                topic: 'component/alerts',
                publishMessage: {
                    jsonMessage: {
                        message: alertData
                    }
                }
            };

            await this.ipcClient.publishToTopic(publishRequest);
```

```
                console.log('Alert sent to AlertHandler component');
            }

        } catch (error) {
            console.error('Error processing sensor data:', error);
        }
    }
}

// Start the component.
const processor = new TemperatureProcessor();
processor.start();
```

C

```
#include <gg/buffer.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/map.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

#define SUBSCRIBE_TOPIC "sensors/temperature"
#define PUBLISH_TOPIC "component/alerts"

typedef struct {
    char sensor_id[64];
    int64_t temperature;
} AlertData;

static pthread_mutex_t alert_mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t alert_cond = PTHREAD_COND_INITIALIZER;
static AlertData pending_alert;
static bool has_pending_alert = false;

static void *alert_publisher_thread(void *arg) {
    (void) arg;
```

```
    while (true) {
        pthread_mutex_lock(&alert_mutex);
        while (!has_pending_alert) {
            pthread_cond_wait(&alert_cond, &alert_mutex);
        }

        AlertData alert = pending_alert;
        has_pending_alert = false;
        pthread_mutex_unlock(&alert_mutex);

        GgBuffer sensor_id_buf = { .data = (uint8_t *)alert.sensor_id, .len =
 strlen(alert.sensor_id) };
        GgMap payload = GG_MAP(
            gg_kv(GG_STR("sensor_id"), gg_obj_buf(sensor_id_buf)),
            gg_kv(GG_STR("temperature"), gg_obj_i64(alert.temperature)),
            gg_kv(GG_STR("alert"), gg_obj_buf(GG_STR("HIGH_TEMPERATURE")))
        );

        GgError ret = ggipc_publish_to_topic_json(GG_STR(PUBLISH_TOPIC), payload);

        if (ret != GG_ERR_OK) {
            fprintf(stderr, "Failed to publish alert\n");
        } else {
            printf("Alert sent to AlertHandler component\n");
        }
    }

    return NULL;
}

static void on_sensor_data(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) topic;
    (void) handle;

    if (gg_obj_type(payload) != GG_TYPE_MAP) {
        fprintf(stderr, "Expected JSON message\n");
        return;
    }

    GgMap map = gg_obj_into_map(payload);
```

```
    GgObject *sensor_id_obj;
    if (!gg_map_get(map, GG_STR("sensor_id"), &sensor_id_obj)) {
        fprintf(stderr, "Missing sensor_id field\n");
        return;
    }
    GgBuffer sensor_id = gg_obj_into_buf(*sensor_id_obj);

    GgObject *temperature_obj;
    if (!gg_map_get(map, GG_STR("temperature"), &temperature_obj)) {
        fprintf(stderr, "Missing temperature field\n");
        return;
    }
    int64_t temperature = gg_obj_into_i64(*temperature_obj);

    printf("Received from sensor %.*s: %lld°F\n",
           (int)sensor_id.len, sensor_id.data, (long long)temperature);

    if (temperature > 80) {
        pthread_mutex_lock(&alert_mutex);
        snprintf(pending_alert.sensor_id, sizeof(pending_alert.sensor_id),
                 "%.*s", (int)sensor_id.len, sensor_id.data);
        pending_alert.temperature = temperature;
        has_pending_alert = true;
        pthread_cond_signal(&alert_cond);
        pthread_mutex_unlock(&alert_mutex);
    }
}

int main(void) {
    setvbuf(stdout, NULL, _IONBF, 0);
    printf("Temperature Processor component starting...\n");

    gg_sdk_init();

    GgError ret = ggipc_connect();
    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to connect to Greengrass nucleus\n");
        exit(1);
    }
    printf("Connected to Greengrass IPC\n");

    // Start alert publisher thread.
    pthread_t alert_thread;
```

```
    if (pthread_create(&alert_thread, NULL, alert_publisher_thread, NULL) != 0) {
        fprintf(stderr, "Failed to create alert publisher thread\n");
        exit(1);
    }

    GgIpcSubscriptionHandle handle;
    ret = ggipc_subscribe_to_topic(
        GG_STR(SUBSCRIBE_TOPIC),
        &on_sensor_data,
        NULL,
        &handle
    );

    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to subscribe to topic\n");
        exit(1);
    }

    printf("Subscribed to %s\n", SUBSCRIBE_TOPIC);
    printf("Waiting for sensor data...\n");

    // Keep running.
    while (true) {
        sleep(1);
    }

    return 0;
}
```

C++

```
#include <gg/ipc/client.hpp>
#include <gg/buffer.hpp>
#include <gg/object.hpp>
#include <gg/types.hpp>

#include <chrono>
#include <condition_variable>
#include <iostream>
#include <mutex>
#include <string>
#include <string_view>
#include <thread>
```

```
struct AlertData {
    std::string sensor_id;
    int64_t temperature;
};

static std::mutex alert_mutex;
static std::condition_variable alert_cv;
static AlertData pending_alert;
static bool has_pending_alert = false;

void alert_publisher_thread() {
    auto& client = gg::ipc::Client::get();

    while (true) {
        std::unique_lock<std::mutex> lock(alert_mutex);
        alert_cv.wait(lock, [] { return has_pending_alert; });

        AlertData alert = pending_alert;
        has_pending_alert = false;
        lock.unlock();

        // Create alert payload as JSON string.
        std::string json_payload = "{\"sensor_id\":\"" + alert.sensor_id +
                                   "\",\"temperature\":" +
 std::to_string(alert.temperature) +
                                   ",\"alert\":\"HIGH_TEMPERATURE\"}";

        // Convert to Buffer and publish.
        gg::Buffer buffer(json_payload);
        auto error = client.publish_to_topic("component/alerts", buffer);

        if (error) {
            std::cerr << "Failed to publish alert\n";
        } else {
            std::cout << "Alert sent to AlertHandler component\n";
        }
    }
}

class SensorCallback : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
```

```
        gg::ipc::Subscription& handle
    ) override {
        (void) topic;
        (void) handle;

        // Payload is a Buffer containing JSON string.
        if (payload.index() != GG_TYPE_BUF) {
            std::cerr << "Expected Buffer message\n";
            return;
        }

        // Extract buffer using gg::get.
        auto buffer = gg::get<std::span<uint8_t>>(payload);
        std::string json_str(reinterpret_cast<const char*>(buffer.data()),
 buffer.size());

        // Simple JSON parsing for demo.
        std::string sensor_id = "sensor1";
        int64_t temperature = 0;

        // Extract temperature (simple string search).
        size_t temp_pos = json_str.find("\"temperature\":");
        if (temp_pos != std::string::npos) {
            temp_pos += 14; // Skip "temperature".
            size_t end_pos = json_str.find_first_of(",}", temp_pos);
            if (end_pos != std::string::npos) {
                temperature = std::stoll(json_str.substr(temp_pos, end_pos -
 temp_pos));
            }
        }

        std::cout << "Received from sensor " << sensor_id << ": "
                  << temperature << "°F\n";

        if (temperature > 80) {
            std::lock_guard<std::mutex> lock(alert_mutex);
            pending_alert = {sensor_id, temperature};
            has_pending_alert = true;
            alert_cv.notify_one();
        }
    }
};

int main() {
```

```
    // Disable stdout buffering for real-time logging.
    std::cout.setf(std::ios::unitbuf);

    std::cout << "Temperature Processor component starting..." << std::endl;

    auto& client = gg::ipc::Client::get();
    std::cout << "Got client instance" << std::endl;

    auto error = client.connect();
    std::cout << "Connect returned, error code: " << error.value() << std::endl;

    if (error) {
        std::cerr << "Failed to connect to Greengrass nucleus: " << error.message()
<< std::endl;
        return 1;
    }

    std::cout << "Connected to Greengrass IPC" << std::endl;

    // Start alert publisher thread.
    std::thread alert_thread(alert_publisher_thread);
    alert_thread.detach();

    // Handler must be static lifetime if subscription handle is not held.
    static SensorCallback handler;
    error = client.subscribe_to_topic("sensors/temperature", handler);

    if (error) {
        std::cerr << "Failed to subscribe to topic: " << error.message() <<
std::endl;
        return 1;
    }

    std::cout << "Subscribed to sensors/temperature" << std::endl;
    std::cout << "Waiting for sensor data..." << std::endl;

    // Keep running.
    while (true) {
        using namespace std::chrono_literals;
        std::this_thread::sleep_for(1s);
    }

    return 0;
```

```
}
```

2. Compila y empaqueta el componente

Algunos lenguajes requieren una compilación o un empaquetado antes de la implementación.

Python

Python no requiere compilación. El componente puede usar el archivo.py directamente.

Java

Para crear un JAR ejecutable con todas las dependencias incluidas:

1. Cree un `pom.xml` archivo en el directorio de su proyecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <!-- Basic project information: organization, component name, and version --
>
    <groupId>com.example</groupId>
    <artifactId>temperature-processor</artifactId>
    <version>1.0.0</version>

    <properties>
        <!-- Java 11 LTS (Long Term Support) is recommended for Greengrass v2
 components -->
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- AWS IoT Device SDK for Java - provides IPC client for Greengrass v2
 local communication -->
        <dependency>
            <groupId>software.amazon.awssdk.iotdevicesdk</groupId>
            <artifactId>aws-iot-device-sdk</artifactId>
            <version>1.25.1</version>
```

```
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Maven Shade Plugin - creates a standalone JAR with all
dependencies included for Greengrass deployment -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <!-- Set the main class for the executable JAR
-->
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>TemperatureProcessor</mainClass>
                                </transformer>
                            </transformers>
                            <filters>
                                <!-- Exclude signature files to avoid security
exceptions -->
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

2. Construye el JAR:

```
mvn clean package
```

   Esto se crea `target/temperature-processor-1.0.0.jar` con todas las dependencias incluidas.

3. Cargue el JAR en su bucket de S3 para su implementación.

JavaScript

Para empaquetar el componente de Node.js con todas las dependencias:

1. Cree un `package.json` archivo:

```
{
  "name": "temperature-processor",
  "version": "1.0.0",
  "description": "Temperature processor component for Greengrass v2",
  "main": "TemperatureProcessor.js",
  "dependencies": {
    "aws-iot-device-sdk-v2": "^1.21.0"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}
```

2. Instale las dependencias en su máquina de desarrollo:

```
npm install
```

   Esto crea una `node_modules` carpeta que contiene la AWS SDK para dispositivos con AWS IoT versión 2.

3. Package para el despliegue:

```
zip -r TemperatureProcessor.zip TemperatureProcessor.js node_modules/
 package.json
```

4. Cargue el archivo zip en su bucket de S3 para el despliegue.

> **Note**
>
> El dispositivo Greengrass debe tener instalado el motor de ejecución Node.js (versión 14 o posterior). No es necesario que lo ejecute `npm install` en el dispositivo principal de Greengrass, ya que el artefacto componente incluye todas las dependencias en la carpeta incluida. `node_modules`

C

Requisitos previos:

Para compilar el SDK y el componente, necesitará las siguientes dependencias de compilación:

- GCC o Clang
- CMake (al menos la versión 3.22)
- Make o Ninja

Instale las dependencias de compilación:

En Ubuntu/Debian:

```
sudo apt install build-essential cmake
```

En Amazon Linux:

```
sudo yum install gcc cmake make
```

Cree un archivo CMake Lists.txt para su componente:

```
cmake_minimum_required(VERSION 3.10)
project(TemperatureProcessor C)

set(CMAKE_C_STANDARD 11)

# Add AWS Greengrass Component SDK
add_subdirectory(aws-greengrass-component-sdk)
```

```
# Build your component executable
add_executable(temperature_processor temperature_processor.c)
target_link_libraries(temperature_processor gg-sdk)
```

Pasos de compilación:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'temperature_processor' is in ./build/
# Upload this binary to S3 for deployment
```

C++

Requisitos previos:

Para compilar el SDK y el componente, necesitarás las siguientes dependencias de compilación:

- GCC o Clang compatibles con C++20
- CMake (al menos la versión 3.22)
- Make o Ninja

Instale las dependencias de compilación:

En Ubuntu/Debian:

```
sudo apt install build-essential cmake
```

En Amazon Linux:

```
sudo yum install gcc-c++ cmake make
```

Cree un archivo CMake Lists.txt para su componente:

```
cmake_minimum_required(VERSION 3.10)
project(TemperatureProcessor CXX)
```

```
set(CMAKE_CXX_STANDARD 20)

# Add SDK as subdirectory
add_subdirectory(aws-greengrass-component-sdk)

# Add C++ SDK subdirectory
add_subdirectory(aws-greengrass-component-sdk/cpp)

add_executable(temperature_processor temperature_processor.cpp)
target_link_libraries(temperature_processor gg-sdk++)
```

Pasos de compilación:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'temperature_processor' will be in ./build/
# Upload this binary to S3 for deployment
```

3. Receta de componentes

Actualice la matriz de «recursos» con los temas reales que utiliza su componente.

Python

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
 AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
```

```
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "component/alerts"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/temperature_processor.py"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/
temperature_processor.py"
        }
      ]
    }
  ]
}
```

Java

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
 AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "component/alerts"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/TemperatureProcessor.jar"
```

```
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/
TemperatureProcessor.jar"
        }
      ]
    }
  ]
}
```

JavaScript

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
 AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "component/alerts"
            ]
          }
```

```
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "cd {artifacts:decompressedPath}/TemperatureProcessor && node
 TemperatureProcessor.js"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/
TemperatureProcessor.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

C/C++

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.TemperatureProcessor",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives sensor data and forwards alerts to
 AlertHandler",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.TemperatureProcessor:pubsub:1": {
            "policyDescription": "Allows access to subscribe to sensor topics",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
```

```
            ],
            "resources": [
              "sensors/temperature"
            ]
          },
          "com.example.TemperatureProcessor:pubsub:2": {
            "policyDescription": "Allows access to publish to alert topics",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "component/alerts"
            ]
          }
        }
      }
    },
    "Manifests": [
      {
        "Platform": {
          "os": "linux",
          "runtime": "*"
        },
        "Lifecycle": {
          "run": "{artifacts:path}/temperature_processor"
        },
        "Artifacts": [
          {
            "Uri": "s3://YOUR-BUCKET/artifacts/com.example.TemperatureProcessor/1.0.0/
temperature_processor"
          }
        ]
      }
    ]
}
```

Escenario 2: Comunicación en la nube

En este escenario se muestra la conversión de una función Lambda de V1 que se comunica con un AWS IoT Core componente genérico de V2.

Arquitectura de aplicaciones

En este ejemplo, se utiliza una arquitectura conectada a la nube:

- AWS IoT Core envía comandos al dispositivo
- La controladora Lambda recibe comandos y los procesa
- La controladora Lambda devuelve los datos de telemetría a AWS IoT Core

Este ejemplo se centra en el controlador Lambda, que muestra cómo recibir mensajes desde y publicar mensajes en él. AWS IoT Core

Suscripciones grupales V1

En AWS IoT Greengrass V1, las siguientes suscripciones grupales permiten la comunicación entre la función Lambda y: AWS IoT Core

Suscripción 1: IoT Cloud → Lambda

- Fuente: IoT Cloud
- Objetivo: Lambda () DeviceController
- Tema: commands/device1

Suscripción 2: Lambda → IoT Cloud

- Fuente: Lambda () DeviceController
- Objetivo: nube de IoT
- Tema: telemetría/dispositivo1

Función Lambda del controlador (V1)

Python

```
import greengrasssdk
import json
import time

iot_client = greengrasssdk.client('iot-data')
```

```
def lambda_handler(event, context):
    """
    Receives commands from IoT Core,
    processes them, and sends telemetry back to cloud
    """
    # Receive command from IoT Core.
    command = event.get('command')
    device_id = event.get('device_id', 'device1')

    print(f"Received command from cloud: {command}")

    # Process command.
    if command == 'get_status':
        status = get_device_status()

        # Send telemetry back to IoT Core.
        telemetry_data = {
            'device_id': device_id,
            'status': status,
            'timestamp': time.time()
        }

        iot_client.publish(
            topic=f'telemetry/{device_id}',
            payload=json.dumps(telemetry_data)
        )

        print(f"Telemetry sent to cloud: {telemetry_data}")

    return {'statusCode': 200}

def get_device_status():
    """Get current device status"""
    # Simulate getting device status.
    return 'online'
```

Java

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.greengrass.javasdk.IotDataClient;
import com.amazonaws.greengrass.javasdk.model.PublishRequest;
import com.google.gson.Gson;
import java.nio.ByteBuffer;
```

```
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;

public class DeviceControllerLambda {
    private static final Gson gson = new Gson();
    private final IotDataClient iotDataClient;

    public DeviceControllerLambda() {
        this.iotDataClient = new IotDataClient();
    }

    public String handleRequest(Map<String, Object> event, Context context) {
        /*
         * Receives commands from IoT Core,
         * processes them, and sends telemetry back to cloud
         */

        // Receive command from IoT Core.
        String command = (String) event.get("command");
        String deviceId = event.containsKey("device_id") ?
            (String) event.get("device_id") : "device1";

        System.out.println("Received command from cloud: " + command);

        // Process command.
        if ("get_status".equals(command)) {
            String status = getDeviceStatus();

            // Send telemetry back to IoT Core.
            Map<String, Object> telemetryData = new HashMap<>();
            telemetryData.put("device_id", deviceId);
            telemetryData.put("status", status);
            telemetryData.put("timestamp", System.currentTimeMillis() / 1000.0);

            String payload = gson.toJson(telemetryData);
            PublishRequest publishRequest = new PublishRequest()
                .withTopic("telemetry/" + deviceId)

 .withPayload(ByteBuffer.wrap(payload.getBytes(StandardCharsets.UTF_8)));

            iotDataClient.publish(publishRequest);

            System.out.println("Telemetry sent to cloud: " + telemetryData);
```

```
        }

        return "Success";
    }

    private String getDeviceStatus() {
        // Simulate getting device status.
        return "online";
    }
}
```

JavaScript

```
const greengrasssdk = require('aws-greengrass-core-sdk');

const iotClient = new greengrasssdk.IotData();

/**
 * Receives commands from IoT Core and sends telemetry back.
 */
exports.handler = function(event, context) {
    console.log('Received command from IoT Core:', JSON.stringify(event));

    const command = event.command;
    const deviceId = event.device_id || 'device1';

    console.log(`Processing command: ${command}`);

    if (command === 'get_status') {
        const status = 'online';

        const telemetryData = {
            device_id: deviceId,
            status: status,
            timestamp: Date.now() / 1000
        };

        // Publish telemetry to IoT Core using greengrasssdk.
        const params = {
            topic: `telemetry/${deviceId}`,
            payload: JSON.stringify(telemetryData)
        };
```

```
        iotClient.publish(params, (err) => {
            if (err) {
                console.error('Error publishing telemetry:', err);
                context.fail(err);
            } else {
                console.log('Telemetry sent to IoT Core:',
 JSON.stringify(telemetryData));
                context.succeed('Success');
            }
        });
    } else {
        context.succeed('Success');
    }
};
```

C

```
#include <aws/greengrass/greengrasssdk.h>
#include <stdio.h>
#include <string.h>
#include <jansson.h>
#include <time.h>

static aws_greengrass_iot_data_client *iot_client = NULL;

const char* get_device_status(void) {
    // Simulate getting device status.
    return "online";
}

void on_cloud_command(const char *topic, const uint8_t *payload, size_t payload_len,
 void *user_data) {
    // Parse incoming command from IoT Core.
    char *payload_str = strndup((char *)payload, payload_len);
    json_error_t error;
    json_t *event = json_loads(payload_str, 0, &error);
    free(payload_str);

    if (!event) {
        fprintf(stderr, "Error parsing JSON: %s\n", error.text);
        return;
    }
```

```
    // Extract command and device_id.
    json_t *command_obj = json_object_get(event, "command");
    json_t *device_id_obj = json_object_get(event, "device_id");

    const char *command = json_string_value(command_obj);
    const char *device_id = device_id_obj ? json_string_value(device_id_obj) :
 "device1";

    printf("Received command from cloud: %s\n", command);

    // Process command.
    if (command && strcmp(command, "get_status") == 0) {
        const char *status = get_device_status();

        // Send telemetry back to IoT Core.
        json_t *telemetry_data = json_object();
        json_object_set_new(telemetry_data, "device_id", json_string(device_id));
        json_object_set_new(telemetry_data, "status", json_string(status));
        json_object_set_new(telemetry_data, "timestamp", json_real(time(NULL)));

        char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

        // Publish telemetry to IoT Core.
        char telemetry_topic[256];
        snprintf(telemetry_topic, sizeof(telemetry_topic), "telemetry/%s",
 device_id);

        aws_greengrass_publish_params params = {
            .topic = telemetry_topic,
            .payload = (uint8_t *)telemetry_payload,
            .payload_len = strlen(telemetry_payload)
        };

        aws_greengrass_iot_data_publish(iot_client, &params);

        printf("Telemetry sent to cloud: %s\n", telemetry_payload);

        free(telemetry_payload);
        json_decref(telemetry_data);
    }

    json_decref(event);
}
```

```
int main(int argc, char *argv[]) {
    // Initialize Greengrass SDK.
    iot_client = aws_greengrass_iot_data_client_new();

    // Subscribe to commands from IoT Core.
    aws_greengrass_subscribe_params subscribe_params = {
        .topic = "commands/device1",
        .callback = on_cloud_command,
        .user_data = NULL
    };

    aws_greengrass_iot_data_subscribe(iot_client, &subscribe_params);

    printf("Device Controller Lambda started\n");
    printf("Subscribed to commands/device1\n");
    printf("Waiting for commands from IoT Core...\n");

    // Keep the Lambda running.
    while (1) {
        sleep(1);
    }

    return 0;
}
```

C++

```
#include <aws/greengrass/greengrasssdk.h>
#include <iostream>
#include <string>
#include <memory>
#include <jansson.h>
#include <ctime>
#include <unistd.h>

class DeviceController {
private:
    std::unique_ptr<aws_greengrass_iot_data_client,
 decltype(&aws_greengrass_iot_data_client_destroy)> iot_client;

    static void message_callback_wrapper(const char *topic, const uint8_t *payload,
                                         size_t payload_len, void *user_data) {
        auto* controller = static_cast<DeviceController*>(user_data);
```

```
        controller->on_cloud_command(topic, payload, payload_len);
    }

    std::string get_device_status() {
        // Simulate getting device status.
        return "online";
    }

public:
    DeviceController()
        : iot_client(aws_greengrass_iot_data_client_new(),
                     aws_greengrass_iot_data_client_destroy) {
        if (!iot_client) {
            throw std::runtime_error("Failed to create Greengrass IoT client");
        }
    }

    void on_cloud_command(const char *topic, const uint8_t *payload, size_t
 payload_len) {
        // Parse incoming command from IoT Core.
        std::string payload_str(reinterpret_cast<const char*>(payload),
 payload_len);

        json_error_t error;
        json_t *event = json_loads(payload_str.c_str(), 0, &error);

        if (!event) {
            std::cerr << "Error parsing JSON: " << error.text << std::endl;
            return;
        }

        // Extract command and device_id.
        json_t *command_obj = json_object_get(event, "command");
        json_t *device_id_obj = json_object_get(event, "device_id");

        const char *command = json_string_value(command_obj);
        const char *device_id = device_id_obj ? json_string_value(device_id_obj) :
 "device1";

        std::cout << "Received command from cloud: " << command << std::endl;

        // Process command.
        if (command && std::string(command) == "get_status") {
            std::string status = get_device_status();
```

```
            // Send telemetry back to IoT Core.
            json_t *telemetry_data = json_object();
            json_object_set_new(telemetry_data, "device_id",
json_string(device_id));
            json_object_set_new(telemetry_data, "status",
json_string(status.c_str()));
            json_object_set_new(telemetry_data, "timestamp",
json_real(std::time(nullptr)));

            char *telemetry_payload = json_dumps(telemetry_data, JSON_COMPACT);

            // Publish telemetry to IoT Core.
            std::string telemetry_topic = "telemetry/" + std::string(device_id);

            aws_greengrass_publish_params params = {
                .topic = telemetry_topic.c_str(),
                .payload = reinterpret_cast<uint8_t*>(telemetry_payload),
                .payload_len = strlen(telemetry_payload)
            };

            aws_greengrass_iot_data_publish(iot_client.get(), &params);

            std::cout << "Telemetry sent to cloud: " << telemetry_payload <<
std::endl;

            free(telemetry_payload);
            json_decref(telemetry_data);
        }

        json_decref(event);
    }

    void subscribe_to_topic(const std::string& topic) {
        aws_greengrass_subscribe_params subscribe_params = {
            .topic = topic.c_str(),
            .callback = message_callback_wrapper,
            .user_data = this
        };

        aws_greengrass_iot_data_subscribe(iot_client.get(), &subscribe_params);

        std::cout << "Device Controller Lambda started" << std::endl;
        std::cout << "Subscribed to " << topic << std::endl;
```

```
        std::cout << "Waiting for commands from IoT Core..." << std::endl;
    }

    void run() {
        // Keep the Lambda running.
        while (true) {
            sleep(1);
        }
    }
};

int main(int argc, char *argv[]) {
    try {
        DeviceController controller;
        controller.subscribe_to_topic("commands/device1");
        controller.run();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Componente genérico (V2)

Para lograr la misma funcionalidad en AWS IoT Greengrass V2, cree un componente genérico con lo siguiente:

1. Código de componente

Python

Requisitos previos: Antes de usar el código de este componente, instale y verifique el código SDK para dispositivos con AWS IoT para Python en su dispositivo Greengrass:

```
# Install the SDK
pip3 install awsiotsdk

# Verify installation
python3 -c "import awsiot.greengrasscoreipc.clientv2; print('SDK installed
 successfully')"
```

Si encuentra conflictos de dependencia durante la instalación, intente instalar una versión específica del. SDK para dispositivos con AWS IoT

Si el comando de verificación muestra «El SDK se instaló correctamente», estás listo para usar el código del componente:

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import QOS
import json
import time

ipc_client = GreengrassCoreIPCClientV2()

def on_command(event):
    """
    Receives commands from IoT Core,
    processes them, and sends telemetry back to cloud
    """
    try:
        # Receive command from IoT Core.
        data = json.loads(event.message.payload.decode('utf-8'))
        command = data.get('command')
        device_id = data.get('device_id', 'device1')

        print(f"Received command from cloud: {command}")

        # Process command.
        if command == 'get_status':
            status = get_device_status()

            # Send telemetry back to IoT Core.
            telemetry_data = {
                'device_id': device_id,
                'status': status,
                'timestamp': time.time()
            }

            ipc_client.publish_to_iot_core(
                topic_name=f'telemetry/{device_id}',
                qos=QOS.AT_LEAST_ONCE,
                payload=json.dumps(telemetry_data).encode('utf-8')
            )
```

```
            print(f"Telemetry sent to cloud: {telemetry_data}")

    except Exception as e:
        print(f"Error processing command: {e}")

def get_device_status():
    """Get current device status"""
    # Simulate getting device status.
    return 'online'

def main():
    print("Device Controller component starting...")

    # Subscribe to commands from IoT Core.
    ipc_client.subscribe_to_iot_core(
        topic_name='commands/device1',
        qos=QOS.AT_LEAST_ONCE,
        on_stream_event=on_command
    )

    print("Subscribed to commands/device1 from IoT Core")
    print("Waiting for commands from cloud...")

    # Keep running.
    while True:
        time.sleep(1)

if __name__ == '__main__':
    main()
```

Java

```
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class DeviceController {
```

```
    private static GreengrassCoreIPCClientV2 ipcClient;

    public static void main(String[] args) {
        System.out.println("Device Controller component starting...");

        try (GreengrassCoreIPCClientV2 client =
GreengrassCoreIPCClientV2.builder().build()) {
            ipcClient = client;

            SubscribeToIoTCoreRequest subscribeRequest = new
SubscribeToIoTCoreRequest()
                .withTopicName("commands/device1")
                .withQos(QOS.AT_LEAST_ONCE);

            ipcClient.subscribeToIoTCore(
                subscribeRequest,
                DeviceController::onCommand,
                Optional.empty(),
                Optional.empty()
            );

            System.out.println("Subscribed to commands/device1 from IoT Core");
            System.out.println("Waiting for commands from cloud...");

            while (true) {
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    public static void onCommand(IoTCoreMessage message) {
        try {
            String payload = new String(message.getMessage().getPayload(),
StandardCharsets.UTF_8);

            // Simple JSON parsing.
            String command = extractJsonValue(payload, "command");
            String deviceId = extractJsonValue(payload, "device_id");
            if (deviceId == null || deviceId.isEmpty()) {
                deviceId = "device1";
            }
```

```
            System.out.println("Received command from cloud: " + command);

            if ("get_status".equals(command)) {
                String status = getDeviceStatus();

                // Build JSON manually.
                String telemetryJson = String.format(
                    "{\"device_id\":\"%s\",\"status\":\"%s\",\"timestamp\":%.3f}",
                    deviceId, status, System.currentTimeMillis() / 1000.0
                );
                byte[] telemetryBytes =
 telemetryJson.getBytes(StandardCharsets.UTF_8);

                PublishToIoTCoreRequest publishRequest = new
 PublishToIoTCoreRequest()
                    .withTopicName("telemetry/" + deviceId)
                    .withQos(QOS.AT_LEAST_ONCE)
                    .withPayload(telemetryBytes);

                ipcClient.publishToIoTCore(publishRequest);

                System.out.println("Telemetry sent to cloud: " + telemetryJson);
            }
        } catch (Exception e) {
            System.err.println("Error processing command: " + e.getMessage());
        }
    }

    private static String extractJsonValue(String json, String key) {
        Pattern pattern = Pattern.compile("\"" + Pattern.quote(key) + "\"\\s*:\\s*
\"([^\"]+)\"");
        Matcher matcher = pattern.matcher(json);
        return matcher.find() ? matcher.group(1) : null;
    }

    private static String getDeviceStatus() {
        return "online";
    }
}
```

JavaScript

```
const greengrasscoreipc = require('aws-iot-device-sdk-v2').greengrasscoreipc;

class DeviceController {
    constructor() {
        this.ipcClient = null;
    }

    async start() {
        console.log('Device Controller component starting...');

        try {
            this.ipcClient = greengrasscoreipc.createClient();
            await this.ipcClient.connect();

            const subscribeRequest = {
                topicName: 'commands/device1',
                qos: 1
            };

            const streamingOperation =
 this.ipcClient.subscribeToIoTCore(subscribeRequest);

            streamingOperation.on('message', (message) => {
                this.onCommand(message);
            });

            streamingOperation.on('streamError', (error) => {
                console.error('Stream error:', error);
            });

            streamingOperation.on('ended', () => {
                console.log('Subscription stream ended');
            });

            await streamingOperation.activate();

            console.log('Subscribed to commands/device1 from IoT Core');
            console.log('Waiting for commands from cloud...');

        } catch (error) {
            console.error('Error starting component:', error);
            process.exit(1);
```

```
        }
    }

    async onCommand(message) {
        try {
            const payload = message.message.payload.toString('utf-8');
            const data = JSON.parse(payload);

            const command = data.command;
            const deviceId = data.device_id || 'device1';

            console.log(`Received command from cloud: ${command}`);

            if (command === 'get_status') {
                const status = this.getDeviceStatus();

                const telemetryData = {
                    device_id: deviceId,
                    status: status,
                    timestamp: Date.now() / 1000
                };

                const telemetryJson = JSON.stringify(telemetryData);

                const publishRequest = {
                    topicName: `telemetry/${deviceId}`,
                    qos: 1,
                    payload: Buffer.from(telemetryJson, 'utf-8')
                };

                await this.ipcClient.publishToIoTCore(publishRequest);
                console.log(`Telemetry sent to cloud: ${telemetryJson}`);
            }

        } catch (error) {
            console.error('Error processing command:', error);
        }
    }

    getDeviceStatus() {
        return 'online';
    }
}
```

```
// Start the component.
const controller = new DeviceController();
controller.start();
```

C

```
#include <gg/buffer.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/map.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <pthread.h>

#define COMMAND_TOPIC "commands/device1"
#define TELEMETRY_TOPIC "telemetry/device1"

typedef struct {
    char device_id[64];
    char command[64];
} CommandData;

static pthread_mutex_t command_mutex = PTHREAD_MUTEX_INITIALIZER;
static pthread_cond_t command_cond = PTHREAD_COND_INITIALIZER;
static CommandData pending_command;
static bool has_pending_command = false;

const char* get_device_status(void) {
    // Simulate getting device status.
    return "online";
}

static void *telemetry_publisher_thread(void *arg) {
    (void) arg;

    while (true) {
        pthread_mutex_lock(&command_mutex);
```

```
        while (!has_pending_command) {
            pthread_cond_wait(&command_cond, &command_mutex);
        }

        CommandData cmd = pending_command;
        has_pending_command = false;
        pthread_mutex_unlock(&command_mutex);

        // Process command.
        if (strcmp(cmd.command, "get_status") == 0) {
            const char *status = get_device_status();

            // Create telemetry JSON string.
            char telemetry_json[512];
            snprintf(telemetry_json, sizeof(telemetry_json),
                     "{\"device_id\":\"%s\",\"status\":\"%s\",\"timestamp\":%ld}",
                     cmd.device_id, status, time(NULL));

            GgBuffer telemetry_buf = {
                .data = (uint8_t *)telemetry_json,
                .len = strlen(telemetry_json)
            };

            // Publish telemetry to IoT Core.
            GgError ret = ggipc_publish_to_iot_core(GG_STR(TELEMETRY_TOPIC),
 telemetry_buf, 0);

            if (ret != GG_ERR_OK) {
                fprintf(stderr, "Failed to publish telemetry to IoT Core\n");
            } else {
                printf("Telemetry sent to cloud: device_id=%s, status=%s\n",
 cmd.device_id, status);
            }
        }
    }

    return NULL;
}

static void on_cloud_command(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) topic;
```

```
    (void) handle;

    printf("Received command from IoT Core: %.*s\n", (int)payload.len,
payload.data);

    // Parse JSON payload (comes as raw buffer from IoT Core).
    // For simplicity, we'll do basic string parsing.

    // Extract command and device_id from JSON string.
    char payload_str[512];
    snprintf(payload_str, sizeof(payload_str), "%.*s", (int)payload.len,
payload.data);

    // Simple JSON parsing (looking for "command":"get_status").
    char *command_start = strstr(payload_str, "\"command\"");
    char *device_id_start = strstr(payload_str, "\"device_id\"");

    if (command_start) {
        pthread_mutex_lock(&command_mutex);

        char *cmd_value = strstr(command_start, ":");
        if (cmd_value) {
            cmd_value = strchr(cmd_value, '"');
            if (cmd_value) {
                cmd_value++;
                char *cmd_end = strchr(cmd_value, '"');
                if (cmd_end) {
                    size_t cmd_len = cmd_end - cmd_value;
                    if (cmd_len < sizeof(pending_command.command)) {
                        strncpy(pending_command.command, cmd_value, cmd_len);
                        pending_command.command[cmd_len] = '\0';
                    }
                }
            }
        }

        // Extract device_id or use default.
        if (device_id_start) {
            char *dev_value = strstr(device_id_start, ":");
            if (dev_value) {
                dev_value = strchr(dev_value, '"');
                if (dev_value) {
                    dev_value++;
                    char *dev_end = strchr(dev_value, '"');
```

```
                    if (dev_end) {
                        size_t dev_len = dev_end - dev_value;
                        if (dev_len < sizeof(pending_command.device_id)) {
                            strncpy(pending_command.device_id, dev_value, dev_len);
                            pending_command.device_id[dev_len] = '\0';
                        }
                    }
                }
            }
        } else {
            strcpy(pending_command.device_id, "device1");
        }

        printf("Received command from cloud: %s\n", pending_command.command);

        has_pending_command = true;
        pthread_cond_signal(&command_cond);
        pthread_mutex_unlock(&command_mutex);
    }
}

int main(void) {
    setvbuf(stdout, NULL, _IONBF, 0);
    printf("Device Controller component starting...\n");

    gg_sdk_init();

    GgError ret = ggipc_connect();
    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to connect to Greengrass nucleus\n");
        exit(1);
    }
    printf("Connected to Greengrass IPC\n");

    // Start telemetry publisher thread.
    pthread_t telemetry_thread;
    if (pthread_create(&telemetry_thread, NULL, telemetry_publisher_thread, NULL) !=
 0) {
        fprintf(stderr, "Failed to create telemetry publisher thread\n");
        exit(1);
    }

    // Subscribe to commands from IoT Core.
    GgIpcSubscriptionHandle handle;
```

```
    ret = ggipc_subscribe_to_iot_core(
        GG_STR(COMMAND_TOPIC),
        0,
        &on_cloud_command,
        NULL,
        &handle
    );

    if (ret != GG_ERR_OK) {
        fprintf(stderr, "Failed to subscribe to IoT Core topic\n");
        exit(1);
    }

    printf("Subscribed to %s\n", COMMAND_TOPIC);
    printf("Waiting for commands from IoT Core...\n");

    while (true) {
        sleep(1);
    }

    return 0;
}
```

C++

```
#include <gg/ipc/client.hpp>
#include <gg/buffer.hpp>
#include <gg/object.hpp>
#include <gg/types.hpp>

#include <chrono>
#include <condition_variable>
#include <ctime>
#include <iostream>
#include <mutex>
#include <string>
#include <string_view>
#include <thread>

constexpr std::string_view COMMAND_TOPIC = "commands/device1";
constexpr std::string_view TELEMETRY_TOPIC = "telemetry/device1";

struct CommandData {
```

```
    std::string device_id;
    std::string command;
};

static std::mutex command_mutex;
static std::condition_variable command_cv;
static CommandData pending_command;
static bool has_pending_command = false;

std::string get_device_status() {
    // Simulate getting device status.
    return "online";
}

void telemetry_publisher_thread() {
    auto& client = gg::ipc::Client::get();

    while (true) {
        std::unique_lock<std::mutex> lock(command_mutex);
        command_cv.wait(lock, [] { return has_pending_command; });

        CommandData cmd = pending_command;
        has_pending_command = false;
        lock.unlock();

        // Process command.
        if (cmd.command == "get_status") {
            std::string status = get_device_status();

            // Get current timestamp.
            auto now = std::time(nullptr);

            // Create telemetry JSON payload.
            std::string telemetry_payload = "{\"device_id\":\"" + cmd.device_id +
                                            "\",\"status\":\"" + status +
                                            "\",\"timestamp\":" + std::to_string(now)
 + "}";

            // Publish telemetry to IoT Core.
            gg::Buffer telemetry_buffer(telemetry_payload);
            auto error = client.publish_to_iot_core(TELEMETRY_TOPIC,
 telemetry_buffer);

            if (error) {
```

```
                std::cerr << "Failed to publish telemetry to IoT Core: "
                          << error.message() << std::endl;
            } else {
                std::cout << "Telemetry sent to cloud: device_id=" << cmd.device_id
                          << ", status=" << status << std::endl;
            }
        }
    }
}

class CloudCommandCallback : public gg::ipc::IoTCoreTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription& handle
    ) override {
        (void) topic;
        (void) handle;

        // Payload is a Buffer containing JSON string from IoT Core.
        if (payload.index() != GG_TYPE_BUF) {
            std::cerr << "Expected Buffer message\n";
            return;
        }

        // Extract buffer.
        auto buffer = gg::get<std::span<uint8_t>>(payload);
        std::string json_str(reinterpret_cast<const char*>(buffer.data()),
 buffer.size());

        std::cout << "Received command from IoT Core: " << json_str << std::endl;

        // Simple JSON parsing for demo.
        std::string command;
        std::string device_id = "device1";  // Default

        // Extract command.
        size_t cmd_pos = json_str.find("\"command\":");
        if (cmd_pos != std::string::npos) {
            size_t start = json_str.find("\"", cmd_pos + 10) + 1;
            size_t end = json_str.find("\"", start);
            if (end != std::string::npos) {
                command = json_str.substr(start, end - start);
            }
```

```
        }

        // Extract device_id if present.
        size_t dev_pos = json_str.find("\"device_id\":");
        if (dev_pos != std::string::npos) {
            size_t start = json_str.find("\"", dev_pos + 12) + 1;
            size_t end = json_str.find("\"", start);
            if (end != std::string::npos) {
                device_id = json_str.substr(start, end - start);
            }
        }

        if (!command.empty()) {
            std::lock_guard<std::mutex> lock(command_mutex);
            pending_command = {device_id, command};
            has_pending_command = true;
            command_cv.notify_one();

            std::cout << "Received command from cloud: " << command << std::endl;
        }
    }
};

int main() {
    // Disable stdout buffering for real-time logging in systemd/Greengrass.
    std::cout.setf(std::ios::unitbuf);

    std::cout << "Device Controller component starting..." << std::endl;

    auto& client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to connect to Greengrass nucleus: "
                  << error.message() << std::endl;
        return 1;
    }

    std::cout << "Connected to Greengrass IPC" << std::endl;

    // Start telemetry publisher thread.
    std::thread telemetry_thread(telemetry_publisher_thread);
    telemetry_thread.detach();
```

```
    // Subscribe to commands from IoT Core.
    static CloudCommandCallback handler;
    error = client.subscribe_to_iot_core(COMMAND_TOPIC, handler);

    if (error) {
        std::cerr << "Failed to subscribe to IoT Core topic: "
                  << error.message() << std::endl;
        return 1;
    }

    std::cout << "Subscribed to " << COMMAND_TOPIC << std::endl;
    std::cout << "Waiting for commands from IoT Core..." << std::endl;

    // Keep running.
    while (true) {
        using namespace std::chrono_literals;
        std::this_thread::sleep_for(1s);
    }

    return 0;
}
```

2. Compila y empaqueta el componente

Algunos lenguajes requieren una compilación o un empaquetado antes de la implementación.

Python

Python no requiere compilación. El componente puede usar el archivo.py directamente.

Java

Para crear un JAR ejecutable con todas las dependencias incluidas:

1. Cree un `pom.xml` archivo en el directorio de su proyecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <!-- Basic project information: organization, component name, and version --
>
    <groupId>com.example</groupId>
    <artifactId>device-controller</artifactId>
    <version>1.0.0</version>

    <properties>
        <!-- Java 11 LTS (Long Term Support) is recommended for Greengrass v2
 components -->
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- AWS IoT Device SDK for Java - provides IPC client for Greengrass v2
 cloud communication -->
        <dependency>
            <groupId>software.amazon.awssdk.iotdevicesdk</groupId>
            <artifactId>aws-iot-device-sdk</artifactId>
            <version>1.25.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Maven Shade Plugin - creates a standalone JAR with all
 dependencies included for Greengrass deployment -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <!-- Set the main class for the executable JAR
 -->
                                <transformer
 implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>DeviceController</mainClass>
```

```
                                </transformer>
                            </transformers>
                            <filters>
                                <!-- Exclude signature files to avoid security
 exceptions -->
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Construye el JAR:

```
mvn clean package
```

   Esto se crea `target/device-controller-1.0.0.jar` con todas las dependencias incluidas.

3. Cargue el JAR en su bucket de S3 para su implementación.

JavaScript

Para empaquetar el componente de Node.js con todas las dependencias:

1. Cree un `package.json` archivo:

```
{
  "name": "device-controller",
  "version": "1.0.0",
  "description": "Device controller component for Greengrass v2",
  "main": "DeviceController.js",
  "dependencies": {
```

```
    "aws-iot-device-sdk-v2": "^1.21.0"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}
```

2. Instale las dependencias en su máquina de desarrollo:

```
npm install
```

Esto crea una `node_modules` carpeta que contiene la AWS SDK para dispositivos con AWS IoT versión 2.

3. Package para el despliegue:

```
zip -r DeviceController.zip DeviceController.js node_modules/ package.json
```

4. Cargue el archivo zip en su bucket de S3 para el despliegue.

> **Note**
>
> El dispositivo Greengrass debe tener instalado el motor de ejecución Node.js (versión 14 o posterior). No es necesario que lo ejecute `npm install` en el dispositivo principal de Greengrass, ya que el artefacto componente incluye todas las dependencias en la carpeta incluida. `node_modules`

C

Requisitos previos:

Para compilar el SDK y el componente, necesitará las siguientes dependencias de compilación:

- GCC o Clang
- CMake (al menos la versión 3.22)
- Make o Ninja

Instale las dependencias de compilación:

En Ubuntu/Debian:

```
sudo apt install build-essential cmake
```

En Amazon Linux:

```
sudo yum install gcc cmake make
```

Cree un archivo CMake Lists.txt para su componente:

```
cmake_minimum_required(VERSION 3.10)
project(DeviceController C)

set(CMAKE_C_STANDARD 11)

# Add AWS Greengrass Component SDK
add_subdirectory(aws-greengrass-component-sdk)

# Build your component executable
add_executable(device_controller device_controller.c)
target_link_libraries(device_controller gg-sdk)
```

Pasos de compilación:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'device_controller' is in ./build/
# Upload this binary to S3 for deployment
```

C++

Requisitos previos:

Para compilar el SDK y el componente, necesitarás las siguientes dependencias de compilación:

- GCC o Clang compatibles con C++20

- CMake (al menos la versión 3.22)
- Make o Ninja

Instale las dependencias de compilación:

En Ubuntu/Debian:

```
sudo apt install build-essential cmake
```

En Amazon Linux:

```
sudo yum install gcc-c++ cmake make
```

Cree un archivo CMake Lists.txt para su componente:

```
cmake_minimum_required(VERSION 3.10)
project(DeviceController CXX)

set(CMAKE_CXX_STANDARD 20)

# Add SDK as subdirectory
add_subdirectory(aws-greengrass-component-sdk)

# Add C++ SDK subdirectory
add_subdirectory(aws-greengrass-component-sdk/cpp)

add_executable(device_controller device_controller.cpp)
target_link_libraries(device_controller gg-sdk++)
```

Pasos de compilación:

```
# Clone the AWS Greengrass Component SDK into your project
git clone https://github.com/aws-greengrass/aws-greengrass-component-sdk.git

# Build your component
cmake -B build -D CMAKE_BUILD_TYPE=MinSizeRel
make -C build -j$(nproc)

# The binary 'device_controller' will be in ./build/
# Upload this binary to S3 for deployment
```

3. Receta de componentes

Actualice la matriz de «recursos» con los temas reales que utiliza su componente.

Python

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/device1"
            ]
          }
        }
      }
    }
  },
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": ">=2.0.0",
      "DependencyType": "HARD"
```

```
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/device_controller.py"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller.py"
        }
      ]
    }
  ]
}
```

Java

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "commands/device1"
            ]
```

```
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "telemetry/device1"
            ]
          }
        }
      }
    }
  },
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": ">=2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/DeviceController.jar"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
DeviceController.jar"
        }
      ]
    }
  ]
}
```

JavaScript

```
{
  "RecipeFormatVersion": "2020-01-25",
```

```
 "ComponentName": "com.example.DeviceController",
 "ComponentVersion": "1.0.0",
 "ComponentType": "aws.greengrass.generic",
 "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
to cloud",
 "ComponentPublisher": "[Your Company]",
 "ComponentConfiguration": {
   "DefaultConfiguration": {
     "accessControl": {
       "aws.greengrass.ipc.mqttproxy": {
         "com.example.DeviceController:mqttproxy:1": {
           "policyDescription": "Allows access to subscribe to command topics from
IoT Core",
           "operations": [
             "aws.greengrass#SubscribeToIoTCore"
           ],
           "resources": [
             "commands/device1"
           ]
         },
         "com.example.DeviceController:mqttproxy:2": {
           "policyDescription": "Allows access to publish telemetry to IoT Core",
           "operations": [
             "aws.greengrass#PublishToIoTCore"
           ],
           "resources": [
             "telemetry/*"
           ]
         }
       }
     }
   }
 },
 "Manifests": [
   {
     "Platform": {
       "os": "linux",
       "runtime": "*"
     },
     "Lifecycle": {
       "run": "cd {artifacts:decompressedPath}/DeviceController && node
DeviceController.js"
     },
     "Artifacts": [
```

```
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
DeviceController.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

C/C++

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.DeviceController",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Receives commands from IoT Core and sends telemetry back
 to cloud",
  "ComponentPublisher": "[Your Company]",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.DeviceController:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to IoT Core topics",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["commands/device1"]
          },
          "com.example.DeviceController:mqttproxy:2": {
            "policyDescription": "Allows access to publish to IoT Core topics",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["telemetry/device1"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
```

```
      },
      "Lifecycle": {
        "run": "{artifacts:path}/device_controller"
      },
      "Artifacts": [
        {
          "Uri": "s3://YOUR-BUCKET/artifacts/com.example.DeviceController/1.0.0/
device_controller"
        }
      ]
    }
  ]
}
```

Actualización de los dispositivos principales de Greengrass V1 a Greengrass V2

Tras comprobar que las aplicaciones y los componentes funcionan en un dispositivo AWS IoT Greengrass V2 principal, puede instalar el software AWS IoT Greengrass principal v2.x en los dispositivos que actualmente ejecutan la versión 1.x, como los dispositivos de producción. A continuación, implemente los componentes de Greengrass V2 para ejecutar las aplicaciones de Greengrass en los dispositivos.

Para actualizar una flota de dispositivos de la V1 a la V2, complete estos pasos para cada dispositivo que desee actualizar. Puede utilizar grupos de objetos para implementar componentes de la V2 en una flota de dispositivos principales.

> **Tip**
>
> Le recomendamos que cree un script para automatizar el proceso de actualización de una flota de dispositivos. Si usaba AWS Systems Manager para administrar su flota, puede usar Systems Manager para ejecutar ese script en cada dispositivo para actualizar su flota de la V1 a la V2.
>
> Puede ponerse en contacto con su representante de AWS Enterprise Support si tiene preguntas sobre la mejor manera de automatizar el proceso de actualización.

Paso 1: Instale la versión AWS IoT Greengrass 2.x del software principal

Elija una de las siguientes opciones para instalar el software AWS IoT Greengrass Core v2.x en un dispositivo básico V1:

- Actualización en menos pasos

  Para realizar la actualización en menos pasos, puede desinstalar el software versión 1.x antes de instalar el software versión 2.x.

- Actualización con un tiempo de inactividad mínimo

  Para realizar la actualización con un tiempo de inactividad mínimo, puede instalar ambas versiones del software AWS IoT Greengrass Core al mismo tiempo. Tras instalar el software AWS IoT Greengrass Core v2.x y comprobar que las aplicaciones de la V2 funcionan correctamente, debe desinstalar el software AWS IoT Greengrass Core v1.x. Antes de elegir esta opción, tenga en cuenta la RAM adicional necesaria para ejecutar ambas versiones del software AWS IoT Greengrass Core al mismo tiempo.

Desinstale AWS IoT Greengrass Core v1.x antes de instalar la v2.x

Si desea realizar la actualización de forma secuencial, desinstale el software AWS IoT Greengrass Core v1.x antes de instalar la v2.x en su dispositivo.

Para desinstalar la versión 1.x del software principal AWS IoT Greengrass

1. Si la AWS IoT Greengrass versión 1.x del software principal se ejecuta como un servicio, debe detener, deshabilitar y eliminar el servicio.

   a. Detenga el servicio AWS IoT Greengrass Core software v1.x en ejecución.

   ```
   sudo systemctl stop greengrass
   ```

   b. Espere hasta que se detenga el servicio. Puede utilizar el comando `list` para comprobar el estado del servicio.

   ```
   sudo systemctl list-units --type=service | grep greengrass
   ```

   c. Deshabilite el servicio.

```
sudo systemctl disable greengrass
```

d. Elimine el servicio.

```
sudo rm /etc/systemd/system/greengrass.service
```

2. Si el software AWS IoT Greengrass Core v1.x no se ejecuta como un servicio, utilice el siguiente comando para detener el daemon. *greengrass-root*Sustitúyalo por el nombre de la carpeta raíz de Greengrass. La ubicación predeterminada es `/greengrass`.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

3. (Opcional) Haga una copia de seguridad de su carpeta raíz de Greengrass y, si corresponde, de su carpeta de escritura personalizada, en una carpeta diferente de su dispositivo.

a. Use el siguiente comando para copiar la carpeta raíz actual de Greengrass en una carpeta diferente y, a continuación, elimine la carpeta raíz.

```
sudo cp -r /greengrass-root /path/to/greengrass-backup
rm -rf /greengrass-root
```

b. Use el siguiente comando para copiar la carpeta de escritura en una carpeta diferente y, a continuación, elimine la carpeta de escritura.

```
sudo cp -r /write-directory /path/to/write-directory-backup
rm -rf /write-directory
```

Para el núcleo de Greengrass: puede utilizar las instrucciones de instalación AWS IoT Greengrass V2 para instalar el núcleo de Greengrass en su dispositivo.

Para Greengrass nucleus lite: Puede utilizar las instrucciones de instalación de Greengrass nucleus lite para instalar Greengrass nucleus lite.

> **Tip**
>
> Para reutilizar la identidad de un dispositivo principal al migrarlo de la V1 a la V2, siga las instrucciones para instalar el software AWS IoT Greengrass Core con aprovisionamiento manual. En primer lugar, elimine el software principal de la versión 1 del dispositivo y, a

continuación, reutilice el componente y el certificado del AWS IoT dispositivo principal de la versión 1 y actualice las AWS IoT políticas del certificado para conceder los permisos que requiere el software de la versión 2.x.

Instale el software AWS IoT Greengrass Core v2.x en un dispositivo que ya ejecute la v1.x

Si instala el software AWS IoT Greengrass Core v2.x en un dispositivo que ya ejecuta el software AWS IoT Greengrass Core v1.x, tenga en cuenta lo siguiente:

- El nombre del dispositivo principal V2 AWS IoT debe ser único. No utilice el mismo nombre de objeto que el de su dispositivo principal V1.
- Los puertos que utilice para la versión 2.x del software AWS IoT Greengrass Core deben ser diferentes de los puertos que utilice para la versión 1.x.
  - Configure el administrador de flujos V1 para que utilice un puerto distinto del 8088. Para obtener más información, consulte la Configuración del administrador de flujos.
  - Configure el agente MQTT de la versión 1 para que utilice un puerto distinto del 8883. Para obtener más información, consulte Configurar el puerto MQTT para la mensajería local.
- AWS IoT Greengrass V2 no ofrece la opción de cambiar el nombre del servicio del sistema Greengrass. Si ejecuta Greengrass como un servicio del sistema, debe realizar una de las siguientes acciones para evitar conflictos en los nombres de los servicios del sistema:
  - Cambie el nombre del servicio de Greengrass en la versión 1.x antes de instalar la versión 2.x.
  - Instale el software AWS IoT Greengrass Core v2.x sin un servicio del sistema y, a continuación, configure manualmente el software como un servicio del sistema con un nombre distinto de. `greengrass`

  Cómo cambiar el nombre del servicio de Greengrass a la versión 1.x

  1. Detenga el servicio AWS IoT Greengrass Core software v1.x.

     ```
     sudo systemctl stop greengrass
     ```

  2. Espere a que se detenga el servicio. El servicio puede tardar unos minutos en detenerse. Puede utilizar el comando `list-units` para comprobar si el servicio se detuvo.

     ```
     sudo systemctl list-units --type=service | grep greengrass
     ```

3. Deshabilite el servicio.

```
sudo systemctl disable greengrass
```

4. Cambie el nombre del servicio.

```
sudo mv /etc/systemd/system/greengrass.service /etc/systemd/system/greengrass-v1.service
```

5. Vuelva a cargar el servicio e inícielo.

```
sudo systemctl daemon-reload
sudo systemctl reset-failed
sudo systemctl enable greengrass-v1
sudo systemctl start greengrass-v1
```

A continuación, puede utilizar las instrucciones de instalación de AWS IoT Greengrass V2 para instalar el software en el dispositivo.

> **Tip**
>
> Para reutilizar la identidad de un dispositivo principal al migrarlo de la versión 1 a la versión 2, siga las instrucciones para instalar el software AWS IoT Greengrass principal con aprovisionamiento manual. En primer lugar, elimine el software principal de la versión 1 del dispositivo y, a continuación, reutilice el componente y el certificado del AWS IoT dispositivo principal de la versión 1 y actualice las AWS IoT políticas del certificado para conceder los permisos que requiere el software de la versión 2.x.

Paso 2: Implemente AWS IoT Greengrass V2 los componentes en los dispositivos principales

Tras instalar la versión 2.x del software AWS IoT Greengrass principal en el dispositivo, implemente los componentes en función del tiempo de ejecución que haya elegido.

Para Greengrass Nucleus:

Cree una implementación que incluya los siguientes recursos. Para implementar componentes en una flota de dispositivos similares, cree una implementación para un grupo de objetos que contenga esos dispositivos.

- Componentes de la función de Lambda que creó a partir de las funciones de Lambda de la V1. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.
- Si utiliza suscripciones V1, el componente enrutador de suscripciones antiguo.
- Si utiliza el administrador de flujos, el componente del administrador de flujos. Para obtener más información, consulte Administración de flujos de datos en los dispositivos principales de Greengrass.
- Si utiliza secretos locales, el componente del administrador de secretos.
- Si utiliza conectores V1, los componentes del conector proporcionados por AWS.
- Si utiliza contenedores de Docker, el componente del administrador de aplicaciones de Docker. Para obtener más información, consulte Ejecución de un contenedor de Docker.
- Si utiliza dispositivos conectados, los componentes compatibles con dispositivos de cliente. También debe habilitar la compatibilidad con los dispositivos de cliente y asociarlos a su dispositivo principal. Para obtener más información, consulte Interacción con dispositivos IoT locales.
- Si utiliza sombras de dispositivo, el componente administrador de sombra. Para obtener más información, consulte Interacción con las sombras de dispositivo.
- Si carga registros de los dispositivos principales de Greengrass a Amazon CloudWatch Logs, el componente del administrador de registros. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.
- Si realiza la integración con AWS IoT SiteWise, siga las instrucciones para configurar el dispositivo principal V2 como AWS IoT SiteWise puerta de enlace. AWS IoT SiteWise proporciona un script de instalación que despliega los AWS IoT SiteWise componentes automáticamente.
- Componentes definidos por el usuario que usted desarrolló para implementar una funcionalidad personalizada.

Para obtener más información sobre cómo crear y revisar implementaciones, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Para Greengrass nucleus lite:

Implemente los componentes genéricos que creó en el paso 2 de la guía de migración en sus dispositivos core lite de Greengrass:

1. Cree sus componentes utilizando las recetas de componentes que ha creado
2. Cree una implementación dirigida a sus dispositivos nucleus lite de Greengrass que incluya sus componentes genéricos
3. Compruebe que sus componentes estén funcionando correctamente

Tutorial: Introducción a AWS IoT Greengrass V2

Puede completar este tutorial de introducción para conocer las características básicas de AWS IoT Greengrass V2. En este tutorial, aprenderá a hacer lo siguiente:

1. Instale y configure el software AWS IoT Greengrass Core en un dispositivo Linux, como un dispositivo Raspberry Pi o Windows. Este dispositivo es un dispositivo principal de Greengrass.
2. Desarrolle un componente Hello World en el dispositivo principal de Greengrass. Los componentes son módulos de software que se ejecutan en dispositivos principales de Greengrass.
3. Cargue ese componente a AWS IoT Greengrass V2 en la Nube de AWS.
4. Implemente ese componente desde la Nube de AWS en el dispositivo principal de Greengrass.

> **Note**
>
> En este tutorial, se describe cómo configurar un entorno de desarrollo y explorar las características de AWS IoT Greengrass. Para obtener más información sobre cómo instalar y configurar los dispositivos de producción, consulte lo siguiente:
>
> - Configuración de los dispositivos AWS IoT Greengrass principales
> - Instalación del software AWS IoT Greengrass Core

Calcule dedicar entre 20 y 30 minutos a este tutorial.

Temas

- Requisitos previos
- Paso 1: configurar una AWS cuenta
- Paso 2: Configurar el entorno
- Paso 3: Instalar el software AWS IoT Greengrass principal
- Paso 4: Desarrollo y prueba de un componente en su dispositivo
- Paso 5: Cree su componente en el AWS IoT Greengrass servicio
- Paso 6: Implementación de su componente
- Pasos a seguir a continuación

Requisitos previos

Para completar este tutorial de introducción, necesita lo siguiente:

- Un Cuenta de AWS. Si no dispone de una, consulte Paso 1: configurar una AWS cuenta.
- Utilice una Región de AWS que admita AWS IoT Greengrass V2. Para ver una lista completa de las regiones compatibles, consulte AWS IoT Greengrass V2 endpoints and quotas en la Referencia general de AWS.
- Un usuario de AWS Identity and Access Management (IAM) con permisos de administrador.
- Un dispositivo para configurar como dispositivo principal de Greengrass, como un dispositivo de Raspberry Pi con sistema operativo Raspberry Pi (anteriormente llamada Raspbian) o un dispositivo Windows 10. Debe tener permisos de administrador en este dispositivo o poder adquirir privilegios de administrador, por ejemplo, a través de `sudo`. Este dispositivo debe tener una conexión a Internet.

  También puede optar por utilizar un dispositivo diferente que cumpla con los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core.

  Si la computadora de desarrollo cumple estos requisitos, puede configurarlo como dispositivo principal de Greengrass en este tutorial.
- Versión 3.5 o posterior de Python instalada para todos los usuarios en el dispositivo y agregada a la variable de entorno `PATH`. En Windows, también debe tener instalado el lanzador de Python para Windows para todos los usuarios.

  Important

  En Windows, Python no se instala para todos los usuarios de forma predeterminada. Al instalar Python, debe personalizar la instalación para configurarla para que el software AWS IoT Greengrass Core ejecute scripts de Python. Por ejemplo, si usa el instalador gráfico de Python, haga lo siguiente:

  1. Elija Instalar el lanzador para todos los usuarios (recomendado).
  2. Elija Customize installation.
  3. Elija Next.
  4. Seleccione Install for all users.
  5. Seleccione Add Python to environment variables.

6. Elija Instalar.

Para obtener más información, consulte Uso de Python en Windows en la documentación de Python 3.

- AWS Command Line Interface (AWS CLI) instalada y configurada con credenciales en la computadora de desarrollo y en el dispositivo. Asegúrese de utilizar la misma Región de AWS para configurar la AWS CLI en la computadora de desarrollo y en el dispositivo. Para usar AWS IoT Greengrass V2 con AWS CLI, debe tener una de las siguientes versiones o las posteriores:
  - Versión mínima de AWS CLI V1: versión 1.18.197
  - Versión mínima de AWS CLI V2: versión 2.1.11

> **Tip**
>
> Puede ejecutar el siguiente comando para comprobar la versión de la AWS CLI que dispone.
>
> ```
> aws --version
> ```

Para obtener más información, consulte Instalar, actualizar y desinstalar la AWS CLI y Configurar la AWS CLI en la Guía del usuario de AWS Command Line Interface.

> **Note**
>
> Si utiliza un dispositivo ARM de 32 bits, como Raspberry Pi con un sistema operativo de 32 bits, instale AWS CLI V1. AWS CLI La versión 2 no está disponible para dispositivos ARM de 32 bits. Para obtener más información, consulte Instalar, actualizar y desinstalar la versión 1 de AWS CLI.

Paso 1: configurar una AWS cuenta

Inscríbase en una Cuenta de AWS

Si no tiene uno Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abrir https://portal.aws.amazon.com/billing/registro.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

   Cuando te registras en un Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar tareas que requieren acceso de usuario raíz.

AWS te envía un correo electrónico de confirmación una vez finalizado el proceso de registro. En cualquier momento, puede ver la actividad de su cuenta actual y administrarla accediendo a https://aws.amazon.com/y seleccionando Mi cuenta.

Creación de un usuario con acceso administrativo

Después de crear un usuario administrativo Cuenta de AWS, asegúrelo Usuario raíz de la cuenta de AWS AWS IAM Identity Center, habilite y cree un usuario administrativo para no usar el usuario root en las tareas diarias.

Proteja su Usuario raíz de la cuenta de AWS

1. Inicie sesión Consola de administración de AWScomo propietario de la cuenta seleccionando el usuario root e introduciendo su dirección de Cuenta de AWS correo electrónico. En la siguiente página, escriba su contraseña.

   Para obtener ayuda para iniciar sesión con el usuario raíz, consulte Iniciar sesión como usuario raíz en la Guía del usuario de AWS Sign-In .
2. Active la autenticación multifactor (MFA) para el usuario raíz.

   Para obtener instrucciones, consulte Habilitar un dispositivo MFA virtual para el usuario Cuenta de AWS raíz (consola) en la Guía del usuario de IAM.

Creación de un usuario con acceso administrativo

1. Activar IAM Identity Center.

Consulte las instrucciones en Activar AWS IAM Identity Center en la Guía del usuario de AWS IAM Identity Center .

2. En IAM Identity Center, conceda acceso administrativo a un usuario.

   Para ver un tutorial sobre su uso Directorio de IAM Identity Center como fuente de identidad, consulte Configurar el acceso de los usuarios con la configuración predeterminada Directorio de IAM Identity Center en la Guía del AWS IAM Identity Center usuario.

Inicio de sesión como usuario con acceso de administrador

- Para iniciar sesión con el usuario de IAM Identity Center, use la URL de inicio de sesión que se envió a la dirección de correo electrónico cuando creó el usuario de IAM Identity Center.

  Para obtener ayuda para iniciar sesión con un usuario del Centro de identidades de IAM, consulte Iniciar sesión en el portal de AWS acceso en la Guía del AWS Sign-In usuario.

Concesión de acceso a usuarios adicionales

1. En IAM Identity Center, cree un conjunto de permisos que siga la práctica recomendada de aplicar permisos de privilegios mínimos.

   Para conocer las instrucciones, consulte Create a permission set en la Guía del usuario de AWS IAM Identity Center .

2. Asigne usuarios a un grupo y, a continuación, asigne el acceso de inicio de sesión único al grupo.

   Para conocer las instrucciones, consulte Add groups en la Guía del usuario de AWS IAM Identity Center .

Paso 2: Configurar el entorno

> **Note**
>
> Estos pasos no son aplicables a la versión lite del núcleo.

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux (Raspberry Pi)

En estos pasos, se supone que utiliza un sistema operativo Raspberry Pi. Si utiliza un dispositivo o sistema operativo diferente, consulte la documentación correspondiente a esto.

Para configurar una Raspberry Pi para AWS IoT Greengrass V2

1. Configure SSH en Raspberry Pi para conectarse de forma remota. Para obtener más información, consulte SSH (Secure Shell) en la documentación de Raspberry Pi.
2. Busque la dirección IP de Raspberry Pi para conectarse mediante SSH. Para ello, puede ejecutar el siguiente comando en Raspberry Pi.

   ```
   hostname -I
   ```

3. Conéctese a Raspberry Pi con SSH.

   En su equipo de desarrollo, ejecute el siguiente comando. *username*Sustitúyalo por el nombre del usuario para iniciar sesión y *pi-ip-address* sustitúyelo por la dirección IP que encontraste en el paso anterior.

   ```
   ssh username@pi-ip-address
   ```

   > **Important**
   >
   > Si su computadora de desarrollo usa una versión anterior de Windows, es posible que no tenga el comando `ssh` o que tenga `ssh`, pero no pueda conectarse a Raspberry Pi. Para conectarse a Raspberry Pi, puede instalar y configurar PuTTY, que es un cliente de SSH de código abierto gratuito. Consulte la documentación de PuTTY para conectarse a Raspberry Pi.

4. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass Core necesita para ejecutarse. En Raspberry Pi, use los siguientes comandos para instalar Java 11.

   ```
   sudo apt install default-jdk
   ```

Cuando la instalación se complete, ejecute el siguiente comando para comprobar que Java se ejecute en Raspberry Pi.

```
java -version
```

El comando imprime la versión de Java que se ejecuta en el dispositivo. El resultado puede tener un aspecto similar al siguiente ejemplo.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

Consejo: Defina los parámetros del núcleo en Raspberry Pi

Si el dispositivo es Raspberry Pi, puede completar los siguientes pasos para ver y actualizar los parámetros del núcleo de Linux:

1. Abra el archivo `/boot/cmdline.txt`. Este archivo especifica los parámetros del núcleo de Linux que se aplicarán cuando arranque la Raspberry Pi.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para abrir el archivo.

   ```
   sudo nano /boot/cmdline.txt
   ```

2. Verifique que el archivo `/boot/cmdline.txt` contenga los siguientes parámetros del núcleo. El parámetro `systemd.unified_cgroup_hierarchy=0` especifica el uso de cgroups v1 en lugar de cgroups versión 2.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Si el archivo `/boot/cmdline.txt` no contiene estos parámetros o los contiene con valores diferentes, actualice el archivo para que contenga estos parámetros y valores.

3. Si actualizó el archivo `/boot/cmdline.txt`, reinicie Raspberry Pi para aplicar los cambios.

```
sudo reboot
```

Configuración de un dispositivo Linux (otro)

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

   ```
   java -version
   ```

   El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   ```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

      ```
      sudo visudo
      ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

      ```
      root    ALL=(ALL:ALL) ALL
      ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:

   a. Pulse la tecla Windows para abrir el menú de inicio.

   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.

   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.

   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.

   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.

   f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.
3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.
4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

   ```
   net user /add ggc_user password
   ```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

   > **Note**
   >
   > En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada

en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Paso 3: Instalar el software AWS IoT Greengrass principal

> **Tip**
>
> Le recomendamos que pruebe el paquete contextual AWS IoT Greengrass AI Agents para configurar y experimentar rápidamente con IoT AWS Greengrass. El paquete contextual para agentes permitirá a los agentes de IA configurar Greengrass Nucleus y Nucleus Lite, implementar componentes y solucionar problemas comunes.

Siga los pasos de esta sección para configurar su Raspberry Pi como un dispositivo AWS IoT Greengrass central que podrá utilizar para el desarrollo local. En esta sección, descargas y ejecutas un instalador que hace lo siguiente para configurar el software AWS IoT Greengrass Core de tu dispositivo:

- Instala el componente núcleo de Greengrass. El núcleo es un componente obligatorio y es el requisito mínimo para ejecutar el software AWS IoT Greengrass Core en un dispositivo. Para obtener más información, consulte Componente de núcleo de Greengrass.
- Registra el dispositivo como una AWS IoT cosa y descarga un certificado digital que permite que el dispositivo se conecte a él AWS. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.
- Agrega la AWS IoT cosa del dispositivo a un grupo de cosas, que es un grupo o una flota de AWS IoT cosas. Los grupos de objetos le permiten administrar flotas de dispositivos principales de Greengrass. Al implementar componentes de software en sus dispositivos, puede optar por implementarlos en dispositivos individuales o en grupos de dispositivos. Para obtener más información, consulte Administración de dispositivos con AWS IoT, en la Guía para desarrolladores de AWS IoT Core .
- Crea el rol de IAM que permite que su dispositivo principal de Greengrass interactúe con servicios de AWS . De forma predeterminada, esta función permite que el dispositivo interactúe con Amazon Logs AWS IoT y envíe registros a Amazon CloudWatch Logs. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

- Instala la interfaz de línea de AWS IoT Greengrass comandos (`greengrass-cli`), que puede usar para probar los componentes personalizados que desarrolle en el dispositivo principal. Para obtener más información, consulte Interfaz de la línea de comandos de Greengrass.

Instalación del software AWS IoT Greengrass Core (consola)

1. Inicie sesión en la consola de AWS IoT Greengrass.
2. En Get started with Greengrass (Comenzar con Greengrass), seleccione Set up core device (Configurar un dispositivo principal).
3. En el Paso 1: Registrar un dispositivo principal de Greengrass, en Nombre del dispositivo principal, introduzca el nombre del objeto AWS IoT para su dispositivo principal de Greengrass. Si el objeto no existe, el instalador la crea.
4. En el Paso 2: Agregar a un grupo de objetos para aplicar una implementación continua, en Grupo de objetos, elija el grupo de objetos AWS IoT al que quiere agregar su dispositivo principal.

   - Si selecciona Introducir un nombre de grupo nuevo, a continuación, en Nombre de grupo de objetos, introduzca el nombre del nuevo grupo que desee crear. El instalador crea el nuevo grupo automáticamente.
   - Si selecciona Seleccionar un grupo existente, luego en Nombre del grupo de objetos, elija el grupo existente que desee usar.
   - Si selecciona Sin grupo, el instalador no agregará el dispositivo principal a un grupo de objetos.
5. En el Paso 3: Instalar el software Greengrass Core, complete los siguientes pasos.

   Nucleus classic

   1. Seleccione Nucleus classic (versión clásica del núcleo) como el tiempo de ejecución del software de su dispositivo principal.
   2. Elija el sistema operativo de su dispositivo principal: Linux o Windows.
   3. Proporcione sus credenciales de AWS al dispositivo para que el instalador pueda aprovisionar los recursos de AWS IoT y de IAM para su dispositivo principal. Para aumentar la seguridad, le recomendamos que obtenga credenciales temporales para un rol de IAM que permita únicamente los permisos mínimos necesarios para el aprovisionamiento. Para obtener más información, consulte Política de IAM mínima para que el instalador aprovisione recursos.

> **Note**
>
> El instalador no guarda ni almacena sus credenciales.

En el dispositivo, realice una de las siguientes acciones para recuperar las credenciales y ponerlas a disposición del instalador del software AWS IoT Greengrass Core:

- (Recomendado) Utilice credenciales temporales de AWS IAM Identity Center

  a. Proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde IAM Identity Center. Para obtener más información, consulte la Actualización manual de credenciales en Obtener y actualizar las credenciales temporales en la Guía del usuario de IAM Identity Center.

  b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass Core.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- Use credenciales de seguridad temporales de un rol de (IAM):

a. Proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde un rol de IAM que asuma. Para obtener más información acerca de cómo recuperar estas credenciales, consulte Solicitud de credenciales de seguridad temporales en la Guía del usuario de IAM.

b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciales a largo plazo de un usuario de IAM:

a. Proporcione el ID de clave de acceso y la clave de acceso secreta del usuario de IAM. Puede crear un usuario de IAM para el aprovisionamiento y luego eliminarlo. Para la política de IAM que debe proporcionarse al usuario, consulte Política de IAM mínima para que el instalador aprovisione recursos. Para obtener información acerca de cómo recuperar credenciales a largo plazo, consulte Administración de las claves de acceso de los usuarios de IAM en la Guía de usuario de IAM.

b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass Core.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
```

c. (Opcional) Si creó un usuario de IAM para aprovisionar su dispositivo de Greengrass, elimínelo.

d. (Opcional) Si utilizó el ID de clave de acceso y la clave de acceso secreta de un usuario de IAM existente, actualice las claves del usuario para que dejen de ser válidas. Para obtener más información, consulte Actualización de claves de acceso en la Guía de usuario de AWS Identity and Access Management.

4. En Ejecutar el instalador, complete los pasos siguientes.

a. En Descargar el instalador, seleccione Copiar y ejecute el comando copiado en su dispositivo principal. Este comando descarga la última versión del software AWS IoT Greengrass Core y la descomprime en el dispositivo.

b. En Ejecutar el instalador, seleccione Copiar y ejecute el comando copiado en su dispositivo principal. Este comando usa los nombres de objetos AWS IoT y grupos de objetos que especificó anteriormente para ejecutar el instalador del software AWS IoT Greengrass Core y configurar los recursos de AWS del dispositivo principal.

Este comando también hace lo siguiente:

- Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

  > **Important**
  >
  > En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass Core como un servicio del sistema.

- Implemente el componente de la CLI de AWS IoT Greengrass, que es una herramienta de línea de comandos que le permite desarrollar componentes personalizados de Greengrass en el dispositivo principal.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.

Al ejecutar este comando, deberían aparecer los siguientes mensajes para indicar que el instalador se ha realizado correctamente.

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> Si tiene un dispositivo Linux y este no tiene systemd, el instalador no configurará el software como un servicio del sistema y no verá el mensaje de éxito al configurar el núcleo como un servicio del sistema.

Nucleus lite

1. Seleccione Nucleus lite (versión lite del núcleo) como el tiempo de ejecución del software de su dispositivo principal.

2. Seleccione el método de configuración de su dispositivo para aprovisionar su dispositivo a un dispositivo principal de Greengrass.

Opción 1: configure un dispositivo con la descarga del paquete (aproximadamente 1 MB)

1. Cree un objeto de AWS IoT y el rol de Greengrass.
2. Descargue el archivo zip que contiene los recursos de AWS IoT a los que su dispositivo necesita conectarse a AWS IoT:
   - Un certificado y una clave privada generados mediante la autoridad de certificación de AWS IoT.
   - Un archivo de esquema para iniciar la instalación de Greengrass en su dispositivo.
3. Descargue el paquete que instalará el tiempo de ejecución más reciente de la versión lite del núcleo de Greengrass en su Raspberry Pi.
4. Aprovisione su dispositivo para que se convierta en un dispositivo de AWS IoT Greengrass Core y conéctelo a AWS IoT:
   a. a. Transfiera el paquete Greengrass y el kit de conexión a su dispositivo mediante una memoria USB, SCP/FTP o tarjetas SD.
   b. b. Descomprima el archivo greengrass-package.zip en el directorio / GreengrassInstaller del dispositivo.
   c. c. Descomprima el archivo zip del kit de conexión en el /directorio del dispositivo.
   d. d. Ejecute el comando proporcionado en el dispositivo para instalar AWS IoT Greengrass
5. A continuación, seleccione Ver dispositivos principales.

Opción 2: configure un dispositivo con una descarga de imagen de muestra de disco completo preconfigurada (aproximadamente 100 MB)

1. Cree un objeto de AWS IoT y el rol de Greengrass.
2. Descargue el archivo zip que contiene los recursos de AWS IoT a los que su dispositivo necesita conectarse a AWS IoT:
   - Un certificado y una clave privada generados mediante la autoridad de certificación de AWS IoT.

- Un archivo de esquema para iniciar la instalación de Greengrass en su dispositivo.

3. Descargue la imagen de muestra de disco completo preconfigurada que contiene Greengrass y el sistema operativo.

   a. Para transferir el kit de conexión y mostrar la imagen en su dispositivo, siga el archivo readme que se descargó con la imagen.

   b. Para iniciar la instalación de Greengrass, encienda y arranque el dispositivo desde la imagen mostrada

4. A continuación, seleccione Ver dispositivos principales.

Opción 3: configure un dispositivo con su propia versión personalizada

1. Cree un objeto de AWS IoT y el rol de Greengrass.
2. Descargue el archivo zip que contiene los recursos de AWS IoT a los que su dispositivo necesita conectarse a AWS IoT:

   - Un certificado y una clave privada generados mediante la autoridad de certificación de AWS IoT.
   - Un archivo de esquema para iniciar la instalación de Greengrass en su dispositivo.

3. Para personalizar y crear su propia imagen con Yocto a partir del código origen y luego usar el kit de conexión para instalar la versión lite del núcleo, siga las instrucciones de GitHub.

   - A continuación, seleccione Ver dispositivos principales.

Instalación del software AWS IoT Greengrass Core (CLI)

> **Note**
>
> Estos pasos no son aplicables en la versión lite del núcleo.

Instalación y configuración del software AWS IoT Greengrass Core

1. En su dispositivo principal de Greengrass, ejecute el siguiente comando para cambiar al directorio de inicio.

Linux or Unix

```
cd ~
```

Windows Command Prompt (CMD)

```
cd %USERPROFILE%
```

PowerShell

```
cd ~
```

2. En su dispositivo principal, descargue el software AWS IoT Greengrass Core a un archivo denominado `greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

3. Descomprima el software AWS IoT Greengrass Core a una carpeta de su dispositivo. Reemplace *`GreengrassInstaller`* por la carpeta que desee usar.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. Ejecute el siguiente comando para iniciar el instalador del software AWS IoT Greengrass Core. Este comando hace lo siguiente:

   - Cree los recursos de AWS que el dispositivo principal necesita para funcionar.
   - Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

     > **Important**
     >
     > En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass Core como un servicio del sistema.

   - Implemente el componente de la CLI de AWS IoT Greengrass, que es una herramienta de línea de comandos que le permite desarrollar componentes personalizados de Greengrass en el dispositivo principal.
   - Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.

   Reemplace los valores de los argumentos en su comando de la siguiente manera.

a. */greengrass/v2* o *C:\greengrass\v2*: la ruta a la carpeta raíz que se usa para instalar el software AWS IoT Greengrass Core.

b. *GreengrassInstaller*. La ruta a la carpeta en la que desempaquetó el instalador de software AWS IoT Greengrass Core.

c. *region*. La Región de AWS en la que se van a buscar o crear recursos.

d. *MyGreengrassCore*. El nombre del objeto AWS IoT para su dispositivo principal de Greengrass. Si el objeto no existe, el instalador la crea. El instalador descarga los certificados para autenticarse como el objeto AWS IoT. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

> **Note**
>
> El nombre del objeto no puede contener dos puntos (`:`).

e. *MyGreengrassCoreGroup*. El nombre del grupo de objetos AWS IoT para su dispositivo principal de Greengrass. Si el grupo de objetos no existe, el instalador lo crea y le agrega un objeto. Si el grupo de objetos existe y tiene una implementación activa, el dispositivo principal descarga y ejecuta el software que especifique la implementación.

> **Note**
>
> El nombre del grupo de objetos no puede contener dos puntos (`:`).

f. *GreengrassV2IoTThingPolicy*. El nombre de la política de AWS IoT que permite a los dispositivos principales de Greengrass comunicarse con AWS IoT y AWS IoT Greengrass. Si la política de AWS IoT no existe, el instalador crea una política de AWS IoT permisiva con este nombre. Puede restringir los permisos de esta política según su caso de uso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

g. *GreengrassV2TokenExchangeRole*. El nombre del rol de IAM que permite al dispositivo principal de Greengrass obtener credenciales de AWS temporales. Si el rol no existe, el instalador lo crea y asocia una política denominada *GreengrassV2TokenExchangeRole*`Access`. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

h. *GreengrassCoreTokenExchangeRoleAlias*. El alias del rol de IAM que permite al dispositivo principal de Greengrass obtener credenciales temporales más adelante. Si el

alias del rol no existe, el instalador lo crea y lo dirige al rol de IAM que especifique. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true \
  --deploy-dev-tools true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true ^
  --deploy-dev-tools true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
```

```
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true `
  --deploy-dev-tools true
```

> **Note**
>
> Si ejecuta AWS IoT Greengrass en un dispositivo con memoria limitada, puede controlar la cantidad de memoria que usa el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

Al ejecutar este comando, deberían aparecer los siguientes mensajes para indicar que el instalador se ha realizado correctamente.

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> Si tiene un dispositivo Linux y este no tiene systemd, el instalador no configurará el software como un servicio del sistema y no verá el mensaje de éxito al configurar el núcleo como un servicio del sistema.

(Opcional) Ejecución del software Greengrass (Linux)

> **Note**
>
> Estos pasos no son aplicables en la versión lite del núcleo.

Si instaló el software como un servicio del sistema, el instalador ejecutará el software por usted. De no ser así, debe ejecutar el software. Para comprobar si el instalador configuró el software como un servicio del sistema, busque la siguiente línea en el resultado del instalador.

```
Successfully set up Nucleus as a system service
```

Si no ve este mensaje, haga lo siguiente para ejecutar el software:

1. Ejecute el siguiente comando para ejecutar el software.

   ```
   sudo /greengrass/v2/alts/current/distro/bin/loader
   ```

   El software imprime el siguiente mensaje si se inicia correctamente.

   ```
   Launched Nucleus successfully.
   ```

2. Debe dejar abierta la consola de comandos actual para que el software AWS IoT Greengrass principal siga funcionando. Si usa SSH para conectarse al dispositivo principal, ejecute el siguiente comando en su computadora de desarrollo para abrir una segunda sesión SSH que pueda usar para ejecutar comandos adicionales en el dispositivo principal. *username*Sustitúyalo por el nombre del usuario para iniciar sesión y *pi-ip-address* sustitúyalo por la dirección IP del dispositivo.

   ```
   ssh username@pi-ip-address
   ```

Para obtener más información acerca de cómo interactuar con estos mensajes en los componentes de Greengrass, consulte Configuración del núcleo de Greengrass como un servicio del sistema.

Verificación de la instalación de la CLI de Greengrass en el dispositivo

> **Note**
>
> Estos pasos no son aplicables en la versión lite del núcleo.

La CLI de Greengrass puede tardar hasta un minuto en implementarse. Ejecute el siguiente comando para verificar el estado de la implementación. *MyGreengrassCore*Sustitúyalo por el nombre del dispositivo principal.

```
aws greengrassv2 list-effective-deployments --core-device-thing-name MyGreengrassCore
```

`coreDeviceExecutionStatus` indica el estado de la implementación en el dispositivo principal. Cuando el estado es `SUCCEEDED`, ejecute el siguiente comando para comprobar que la CLI de Greengrass está instalada y en funcionamiento. Sustituya */greengrass/v2* por la ruta a la carpeta raíz.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli help
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli help
```

El comando genera información de ayuda para la CLI de Greengrass. Si no encuentra `greengrass-cli`, es posible que la implementación no haya podido instalar la CLI de Greengrass. Para obtener más información, consulte Solución de problemas AWS IoT Greengrass V2.

También puede ejecutar el siguiente comando para implementar manualmente la AWS IoT Greengrass CLI en su dispositivo.

- *region*Sustitúyala por la Región de AWS que utilices. Asegúrese de usar el mismo Región de AWS que usó para configurar AWS CLI el dispositivo.
- *account-id*Sustitúyala por tu Cuenta de AWS ID.
- *MyGreengrassCore*Reemplácelo por el nombre de su dispositivo principal.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
```

```
      "componentVersion": "2.16.1"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.16.1\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.16.1\"}}'
```

> **Tip**
>
> Puede agregar */greengrass/v2*/bin (Linux) o *C:\greengrass\v2*\bin (Windows) a su variable de entorno PATH para que se ejecute `greengrass-cli` sin su ruta absoluta.

El software AWS IoT Greengrass principal y las herramientas de desarrollo local se ejecutan en su dispositivo. A continuación, puede desarrollar un AWS IoT Greengrass componente de Hello World en su dispositivo.

Paso 4: Desarrollo y prueba de un componente en su dispositivo

Un componente es un módulo de software que se ejecuta en los dispositivos AWS IoT Greengrass principales. Los componentes le permiten crear y administrar aplicaciones complejas como bloques de compilación discretos que puede reutilizar de un dispositivo principal de Greengrass a otro. Cada componente incluye una receta y artefactos.

- Recetas

  Cada componente contiene un archivo de recetas que define sus metadatos. La receta también especifica los parámetros de configuración, dependencias del componente, ciclo de vida y compatibilidad de plataforma del componente. El ciclo de vida del componente define los

comandos que instalan, ejecutan y apagan el componente. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.

Puede definir recetas en formato JSON o YAML.

- Artefactos

  Los componentes pueden tener cualquier número de artefactos, que son componentes binarios. Los artefactos pueden incluir scripts, código compilado, recursos estáticos y cualquier otro archivo que utilice un componente. Los componentes también pueden utilizar artefactos de las dependencias de los componentes.

Con AWS IoT Greengrass, puede usar la CLI de Greengrass para desarrollar y probar componentes localmente en un dispositivo central de Greengrass sin interactuar con la nube. AWS Cuando complete su componente local, podrá usar la receta y los artefactos del componente para crear ese componente en el AWS IoT Greengrass servicio en la AWS nube y, a continuación, implementarlo en todos sus dispositivos principales de Greengrass. Para obtener más información sobre los componentes, consulte Desarrollo de componentes de AWS IoT Greengrass.

En esta sección, aprenderá a crear y ejecutar un componente básico de Hello World de forma local en su dispositivo principal.

Desarrollo de un componente de Hello World en su dispositivo

1. Cree una carpeta para sus componentes con subcarpetas para recetas y artefactos. Ejecute los siguientes comandos en su dispositivo principal de Greengrass para crear estas carpetas y cambiarlas a la carpeta de componentes. Sustituya *~/greengrassv2* o *%USERPROFILE%\greengrassv2* por la ruta a la carpeta que desee utilizar para el desarrollo local.

   Linux or Unix

   ```
   mkdir -p ~/greengrassv2/{recipes,artifacts}
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
   cd %USERPROFILE%\greengrassv2
   ```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. Use un editor de texto para crear un archivo de recetas que define los metadatos, parámetros, dependencias, ciclo de vida y capacidad de plataforma de su componente. Incluya la versión del componente en el nombre del archivo de recetas para poder identificar qué receta refleja qué versión del componente. Puede elegir el formato YAML o JSON para su receta.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   JSON

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   nano recipes/com.example.HelloWorld-1.0.0.yaml
   ```

   > **Note**
   >
   > AWS IoT Greengrass usa versiones semánticas para los componentes. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. Por ejemplo, la versión `1.0.0` representa el primer lanzamiento principal de un componente. Para obtener más información, consulte la especificación semántica de la versión.

3. Pegue la siguiente receta en el archivo.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.HelloWorld",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "My first AWS IoT Greengrass component.",
     "ComponentPublisher": "Amazon",
   ```

```
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
```

```
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

La receta de la sección `ComponentConfiguration` define un parámetro, `Message`, cuyo valor predeterminado es `world`. La sección `Manifests` define un manifiesto, que es un conjunto de instrucciones y artefactos del ciclo de vida de una plataforma. Puede definir varios manifiestos para especificar diferentes instrucciones de instalación para distintas plataformas, por ejemplo. En el manifiesto, la sección `Lifecycle` indica al dispositivo principal de Greengrass que ejecute el script Hello World con el valor del parámetro `Message` como un argumento.

4. Ejecute el siguiente comando para crear una carpeta para los artefactos del componente.

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir artifacts\com.example.HelloWorld\1.0.0
   ```

   PowerShell

   ```
   mkdir artifacts\com.example.HelloWorld\1.0.0
   ```

   Important

   Debe usar el siguiente formato para la ruta de la carpeta de artefactos. Incluya el nombre y la versión del componente que especifique en la receta.

   ```
   artifacts/componentName/componentVersion/
   ```

5. Use un editor de texto para crear un archivo de artefactos de script de Python para su componente Hello World.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Copie y pegue el siguiente script de Python en el archivo.

```
import sys

message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

6. Utilice la AWS IoT Greengrass CLI local para gestionar los componentes de su dispositivo principal de Greengrass.

   Ejecute el siguiente comando para implementar el componente en el AWS IoT Greengrass núcleo. Sustituya */greengrass/v2* o *C:\greengrass\v2* por AWS IoT Greengrass V2 la carpeta raíz y sustituya *~/greengrassv2* o *%USERPROFILE%\greengrassv2* por la carpeta de desarrollo de componentes.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
   ```

```
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

Este comando agrega el componente que usa la receta en `recipes` y el script de Python en `artifacts`. La opción `--merge` agrega o actualiza el componente y la versión que especifique.

7. El software AWS IoT Greengrass Core guarda la salida estándar del proceso de los componentes en los archivos de registro de la `logs` carpeta. Ejecute el siguiente comando para comprobar que el componente Hello World ejecuta e imprime mensajes.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.HelloWorld.log
   ```

   El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
   ```

   Debería ver mensajes similares al del siguiente ejemplo.

   ```
   Hello, world!
   ```

   > **Note**
   >
   > Si el archivo no existe, es posible que la implementación local aún no esté completa. Si el archivo no existe en 15 segundos, es probable que la implementación haya fallado. Esto puede ocurrir si la receta no es válida, por ejemplo. Ejecute el siguiente comando para ver el archivo de registro AWS IoT Greengrass principal. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.log
```

El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

8. Modifique el componente local para iterar y probar el código. Ábrelo `hello_world.py` en un editor de texto y añade el siguiente código en la línea 4 para editar el mensaje que registra el AWS IoT Greengrass núcleo.

```
message += " Greetings from your first Greengrass component."
```

Ahora, el script `hello_world.py` debería tener los siguientes contenidos.

```
import sys

message = "Hello, %s!" % sys.argv[1]
message += " Greetings from your first Greengrass component."

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

9. Ejecute el siguiente comando para actualizar el componente con los cambios.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.example.HelloWorld=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

Este comando actualiza el componente `com.example.HelloWorld` con el último artefacto de Hello World.

10. Ejecute el siguiente comando para reiniciar el componente. Al reiniciar un componente, el dispositivo principal usa los cambios más recientes.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli component restart \
      --names "com.example.HelloWorld"
    ```

    Windows Command Prompt (CMD)

    ```
    C:\greengrass\v2\bin\greengrass-cli component restart ^
      --names "com.example.HelloWorld"
    ```

    PowerShell

    ```
    C:\greengrass\v2\bin\greengrass-cli component restart `
      --names "com.example.HelloWorld"
    ```

11. Vuelva a comprobar el registro para verificar que el componente Hello World imprime el nuevo mensaje.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

Debería ver mensajes similares al del siguiente ejemplo.

```
Hello, world! Greetings from your first Greengrass component.
```

12. Puede actualizar los parámetros de configuración del componente para probar diferentes configuraciones. Al implementar un componente, puede especificar una actualización de configuración, que defina cómo modificar la configuración del componente en el dispositivo principal. Puede especificar qué valores de configuración desea restablecer a los valores predeterminados y los nuevos valores de configuración que se van a fusionar en el dispositivo principal. Para obtener más información, consulte Actualización de las configuraciones de los componentes.

    Haga lo siguiente:

    a. Use un editor de texto para crear un archivo llamado `hello-world-config-update.json` para contener la actualización de configuración

       Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

       ```
       nano hello-world-config-update.json
       ```

b. Copie y pegue el siguiente objeto JSON en el archivo. Este objeto JSON define una actualización de configuración que combina el valor `friend` con el parámetro `Message` para actualizar su valor. Esta actualización de configuración no especifica ningún valor que se vaya a restablecer. No es necesario restablecer el parámetro `Message` porque la actualización de la combinación reemplaza el valor existente.

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "Message": "friend"
    }
  }
}
```

c. Ejecute el siguiente comando para implementar la actualización de configuración en el componente Hello World.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-config-update.json
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --merge "com.example.HelloWorld=1.0.0" ^
  --update-config hello-world-config-update.json
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --merge "com.example.HelloWorld=1.0.0" `
  --update-config hello-world-config-update.json
```

d. Vuelva a comprobar el registro para comprobar que el componente Hello World genera el nuevo mensaje.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

Debería ver mensajes similares al del siguiente ejemplo.

```
Hello, friend! Greetings from your first Greengrass component.
```

13. Cuando termine de probar el componente, retírelo del dispositivo principal. Ejecute el comando siguiente.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli deployment create --
    remove="com.example.HelloWorld"
    ```

    Windows Command Prompt (CMD)

    ```
    C:\greengrass\v2\bin\greengrass-cli deployment create --
    remove="com.example.HelloWorld"
    ```

    PowerShell

    ```
    C:\greengrass\v2\bin\greengrass-cli deployment create --
    remove="com.example.HelloWorld"
    ```

> **Important**
>
> Este paso es necesario para volver a implementar el componente en el dispositivo principal después de cargarlo en AWS IoT Greengrass. De lo contrario, se produce un error en la implementación y se produce un error de compatibilidad de versiones porque la implementación local especifica una versión diferente del componente.

Ejecute el siguiente comando y compruebe que el componente `com.example.HelloWorld` no aparezca en la lista de componentes del dispositivo.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component list
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component list
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component list
```

Su componente Hello World está completo y ahora puede subirlo al servicio AWS IoT Greengrass en la nube. A continuación, puede implementar el componente en los dispositivos principales de Greengrass.

Paso 5: Cree su componente en el AWS IoT Greengrass servicio

Cuando termine de desarrollar un componente en su dispositivo principal, puede cargarlo en el servicio de AWS IoT Greengrass en Nube de AWS. También puede crear el componente directamente en la consola de AWS IoT Greengrass. AWS IoT Greengrass proporciona un servicio de administración de componentes que aloja sus componentes para que pueda implementarlos en dispositivos individuales o en flotas de dispositivos. Para cargar un componente en el AWS IoT Greengrass servicio, debe completar los siguientes pasos:

- Carga de artefactos de componentes en un bucket de S3.
- Agregue el URI de Amazon Simple Storage Service (Amazon S3) de cada artefacto a la receta del componente.
- Cree un componente AWS IoT Greengrass a partir de la receta del componente.

En esta sección, debe completar estos pasos en su dispositivo principal de Greengrass para cargar su componente Hello World en el AWS IoT Greengrass servicio.

Cree su componente en AWS IoT Greengrass (consola)

1. Utilice un depósito de S3 en su AWS cuenta para alojar los artefactos de los AWS IoT Greengrass componentes. Al implementar el componente en un dispositivo principal, el dispositivo descarga los artefactos del componente del bucket.

   Puede crear un nuevo bucket de S3 o utilizar un bucket existente.

   a. En la consola de Amazon S3, en Buckets, elija Crear bucket.

   b. En Nombre del bucket, escriba un nombre único para el bucket. Por ejemplo, puede utilizar **greengrass-component-artifacts-*region*-*123456789012***. *123456789012*Sustitúyalo por tu ID *region* de AWS cuenta y por el Región de AWS que utilices para este tutorial.

   c. Para AWS la región, selecciona la AWS región que utilizas para este tutorial.

   d. Elija Crear bucket.

   e. En Buckets, elija el bucket que ha creado y cargue el script `hello_world.py` en la carpeta `artifacts/com.example.HelloWorld/1.0.0` del bucket. Para obtener más información acerca de cómo cargar objetos a buckets de S3, consulte Carga de objetos en la Guía del usuario de Amazon Simple Storage Service.

   f. Copie el URI de S3 del objeto `hello_world.py` en el bucket de S3. Este URI debería parecerse al siguiente ejemplo. Sustituya amzn-s3-demo-bucket por el nombre de su bucket de S3.

   ```
   s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

2. Permita que el dispositivo principal acceda a los artefactos de los componentes del bucket de S3.

Cada dispositivo principal tiene una función de IAM del dispositivo principal que le permite interactuar con los registros AWS IoT y enviarlos a la AWS nube. Este rol de dispositivo no permite el acceso a los bucket de S3 de forma predeterminada, por lo que debe crear y adjuntar una política que permita que el dispositivo principal recupere artefactos de componentes del bucket de S3.

Puede omitir este paso si el rol de su dispositivo ya permite acceder al bucket de S3. De lo contrario, cree una política de IAM que permita el acceso y adjúntela al rol, de la siguiente manera:

a. En el panel de navegación de la consola de IAM, elija Políticas, seguido de Crear política.

b. En la pestaña JSON, reemplace el contenido del marcador de posición por la política siguiente. Reemplace amzn-s3-demo-bucket por el nombre del bucket de S3 que contiene los artefactos de los componentes para que los descargue el dispositivo principal.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

c. Elija Siguiente.

d. En la sección Detalles de política, en Nombre, introduzca **`MyGreengrassV2ComponentArtifactPolicy`**.

e. Elija Crear política.

f. En el menú de navegación de la consola de IAM, seleccione Rol y, a continuación, elija el nombre del rol para el dispositivo principal. Especificó este nombre de función al instalar el software AWS IoT Greengrass principal. Si no especifica un nombre, el nombre predeterminado es `GreengrassV2TokenExchangeRole`.

g. En la pestaña Permisos, elija Agregar permisos y, a continuación, Asociar políticas.

h. En la sección Otras políticas de permisos, seleccione la casilla de verificación junto a la política `MyGreengrassV2ComponentArtifactPolicy` que creó, a continuación, elija Agregar permisos.

3. Utilice la receta del componente para crear un componente en la consola de AWS IoT Greengrass.

   a. En el panel de navegación de la consola de AWS IoT Greengrass, elija Componentes, seguido de Crear componente.

   b. En Información del componente, seleccione Introducir receta como JSON. La receta del marcador de posición debería verse similar al siguiente ejemplo.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "Run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
```

```
      },
      "Lifecycle": {
        "Run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

c. Sustituya el URI del marcador de posición de cada sección `Artifacts` por el URI de S3 de su objeto `hello_world.py`.

d. Seleccione Crear componente.

e. En com.example. HelloWorlden la página del componente, compruebe que el estado del componente es desplegable.

Cree su componente en AWS IoT Greengrass ()AWS CLI

Cómo cargar su componente Hello World

1. Utilice un depósito de S3 en el suyo Cuenta de AWS para alojar los artefactos de los AWS IoT Greengrass componentes. Al implementar el componente en un dispositivo principal, el dispositivo descarga los artefactos del componente del bucket.

   Puede crear un bucket de S3 existente o ejecutar el siguiente comando para crear un bucket. Este comando crea un depósito con su Cuenta de AWS ID y Región de AWS forma un nombre de depósito único. *123456789012*Sustitúyalo por tu Cuenta de AWS ID y *region* por el Región de AWS que utilices para este tutorial.

   ```
   aws s3 mb s3://greengrass-component-artifacts-123456789012-region
   ```

   El comando genera la siguiente información si la solicitud tiene éxito.

   ```
   make_bucket: greengrass-component-artifacts-123456789012-region
   ```

2. Permita que el dispositivo principal acceda a los artefactos de los componentes del bucket de S3.

   Cada dispositivo principal tiene una función de IAM del dispositivo principal que le permite interactuar con los registros AWS IoT y enviarlos. Nube de AWS Este rol de dispositivo no permite el acceso a los bucket de S3 de forma predeterminada, por lo que debe crear y adjuntar una política que permita que el dispositivo principal recupere artefactos de componentes del bucket de S3.

   Puede omitir este paso si el rol del dispositivo principal ya permite acceder al bucket de S3. De lo contrario, cree una política de IAM que permita el acceso y adjúntela al rol, de la siguiente manera:

   a. Cree un archivo llamado `component-artifact-policy.json` y copie el siguiente JSON en el archivo. Esta política permite el acceso a todos los archivos en un bucket de S3. Sustituya amzn-s3-demo-bucket por el nombre de su bucket de S3.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
          }
        ]
      }
      ```

   b. Ejecute el siguiente comando para crear la política del documento de política en `component-artifact-policy.json`.

      Linux or Unix

      ```
      aws iam create-policy \\
        --policy-name MyGreengrassV2ComponentArtifactPolicy \\
        --policy-document file://component-artifact-policy.json
      ```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name MyGreengrassV2ComponentArtifactPolicy ^
  --policy-document file://component-artifact-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name MyGreengrassV2ComponentArtifactPolicy `
  --policy-document file://component-artifact-policy.json
```

Copie la política del nombre de recurso de Amazon (ARN) de la política de los metadatos de salida. Utilice este ARN para asociar la política al rol del dispositivo principal en el siguiente paso.

c. Ejecute el siguiente comando para asociar la política al rol del dispositivo principal. *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre de la función del dispositivo principal. Especificó este nombre de función al instalar el software AWS IoT Greengrass Core. Sustituya el ARN de la política por el ARN del paso anterior.

Linux or Unix

```
aws iam attach-role-policy \\
  --role-name GreengrassV2TokenExchangeRole \\
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
```

```
  --policy-arn
arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Si todo es correcto, el comando no tiene salida. El dispositivo principal ahora puede acceder a los artefactos que carga en este bucket de S3.

3. Cargue el artefacto de script de Python de Hello World en el bucket de S3.

   Ejecute el siguiente comando para cargar el script en la misma ruta del depósito en el que se encuentra el script en su AWS IoT Greengrass núcleo. Sustituya amzn-s3-demo-bucket por el nombre de su bucket de S3.

   Linux or Unix

   ```
   aws s3 cp \
     artifacts/com.example.HelloWorld/1.0.0/hello_world.py \
     s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   Windows Command Prompt (CMD)

   ```
   aws s3 cp ^
     artifacts/com.example.HelloWorld/1.0.0/hello_world.py ^
     s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   PowerShell

   ```
   aws s3 cp `
     artifacts/com.example.HelloWorld/1.0.0/hello_world.py `
     s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   El comando genera una línea que comienza con `upload:` si la solicitud se realiza correctamente.

4. Agregue el URI de Amazon S3 de cada artefacto a la receta del componente.

   El URI de Amazon S3 está compuesto por el nombre del bucket y la ruta al objeto artefacto del bucket. El URI de Amazon S3 de su artefacto de script es el URI en el que cargó el artefacto en el paso anterior. Este URI debería parecerse al siguiente ejemplo. Sustituya amzn-s3-demo-bucket por el nombre de su bucket de S3.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

Para agregar el artefacto a la receta, agregue una lista de `Artifacts` que contenga una estructura con la URI de Amazon S3.

JSON

```
"Artifacts": [
  {
    "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py"
  }
]
```

Abra el archivo de receta en un editor de texto.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

Agregue el artefacto a la receta. El archivo de receta debe ser similar al siguiente ejemplo.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
```

```
        "Run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "Run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
Artifacts:
  - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

Abra el archivo de receta en un editor de texto.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

Agregue el artefacto a la receta. El archivo de receta debe ser similar al siguiente ejemplo.

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      Run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      Run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

5. Crea un recurso componente a AWS IoT Greengrass partir de la receta. Ejecute el siguiente comando para crear el componente a partir de la receta, que proporciona como un archivo binario.

   JSON

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.yaml
   ```

Si la solicitud se realiza con éxito, la respuesta de es similar a la del siguiente ejemplo.

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
  "componentName": "com.example.HelloWorld",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Nov 30 09:04:05 UTC 2020",
  "status": {
    "componentState": "REQUESTED",
    "message": "NONE",
    "errors": {}
  }
}
```

Copie el `arn` de la respuesta para comprobar el estado del componente en el paso siguiente.

> **Note**
>
> También puede ver su componente Hello World en la consola de AWS IoT Greengrass, en la página Componentes.

6. Compruebe que el componente se ha creado y está listo para ser implementado. Al crear un componente, su estado es `REQUESTED`. A continuación, AWS IoT Greengrass valida que el componente sea desplegable. Puede ejecutar el siguiente comando para consultar el estado del componente y comprobar que el componente se puede implementar. Sustituya `arn` por el ARN del paso anterior.

```
aws greengrassv2 describe-component --arn
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0"
```

Si el componente se valida, la respuesta indica que el estado del componente es `DEPLOYABLE`.

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
  "componentName": "com.example.HelloWorld",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-11-30T18:04:05.823Z",
```

```
  "publisher": "Amazon",
  "description": "My first Greengrass component.",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "os": "linux",
      "architecture": "all"
    }
  ]
}
```

Su componente Hello World ya está disponible en. AWS IoT Greengrass Puede volver a implementarlo en este dispositivo principal de Greengrass o en otros dispositivos principales.

Paso 6: Implementación de su componente

Con AWS IoT Greengrass, puede implementar componentes en dispositivos individuales o grupos de dispositivos. Al implementar un componente, AWS IoT Greengrass instala y ejecuta el software de ese componente en cada dispositivo de destino. Usted especifica los componentes que se van a implementar y la actualización de configuración que se va a implementar para cada componente. También puede controlar cómo se lleva a cabo la implementación en los dispositivos a los que se dirige la implementación. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

En esta sección, vuelva a implementar el componente Hello World en su dispositivo principal de Greengrass.

Implementación de su componente (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, en la pestaña Mis componentes, elija com.example.HelloWorld.
3. En la página com.example.HelloWorld, elija Implementar.
4. En Agregar a la implementación, elija Crear nueva implementación y, a continuación, elija Siguiente.

5. En la página Especificar detalles, haga lo siguiente:

   a. En el cuadro Name (Nombre), introduzca **`Deployment for MyGreengrassCore`**.

   b. Para el Destino de la implementación, elija Dispositivo principal y el nombre del objeto de AWS IoT del dispositivo principal. El valor predeterminado de este tutorial es *`MyGreengrassCore`*.

   c. Elija Siguiente.

6. En la página Seleccionar componentes, en Mis componentes, compruebe que el componente com.example.HelloWorld esté seleccionado y elija Siguiente.

7. En la página Configurar componentes, elija com.example.HelloWorld y haga lo siguiente:

   a. Seleccione Configurar componente.

   b. En Actualización de la configuración, en Configuración para fusionar, introduzca la siguiente configuración.

   ```
   {
     "Message": "universe"
   }
   ```

   Esta actualización de configuración establece el parámetro `Message` de Hello World en `universe` para el dispositivo de esta implementación.

   c. Seleccione Confirmar.

   d. Elija Siguiente.

8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.

9. En la página Revisar, elija Implementar.

10. Compruebe que la implementación se complete correctamente. La implementación puede tardar varios minutos en completarse. Consulte el registro de Hello World para comprobar el cambio. Ejecute el siguiente comando en su dispositivo principal de Greengrass.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
```

Debería ver mensajes similares al del siguiente ejemplo.

```
Hello, universe! Greetings from your first Greengrass component.
```

> **Note**
>
> Si los mensajes de registro no cambian, la implementación ha fallado o no ha llegado al dispositivo principal. Esto puede ocurrir si el dispositivo principal no está conectado a Internet o no tiene permisos para recuperar artefactos del bucket de S3. Ejecute el siguiente comando en su dispositivo principal para ver el archivo de registro del software AWS IoT Greengrass Core. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```

Para obtener más información, consulte Solución de problemas AWS IoT Greengrass V2.

Implementación de su componente (AWS CLI)

Implementación de su componente Hello World

1. En su computadora de desarrollo, cree un archivo llamado `hello-world-deployment.json` y copie el siguiente JSON en el archivo. Este archivo define los componentes y las configuraciones que se van a implementar.

```
{
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"Message\":\"universe\"}"
      }
    }
  }
}
```

   Este archivo de configuración especifica que se debe implementar la versión `1.0.0` del componente Hello World que desarrolló y publicó en el procedimiento anterior. La `configurationUpdate` especifica combinar la configuración del componente en una cadena codificada en JSON. Esta actualización de configuración establece el parámetro `Message` de Hello World en `universe` para el dispositivo de esta implementación.

2. Ejecute el siguiente comando para implementar el componente de su dispositivo principal de Greengrass. Puede realizar la implementación en objetos, que son dispositivos individuales, o en grupos de objetos, que son grupos de dispositivos. Reemplace *`MyGreengrassCore`* por el nombre del objeto AWS IoT del dispositivo principal.

   Linux or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
```

```
  --cli-input-json file://hello-world-deployment.json
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --cli-input-json file://hello-world-deployment.json
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --cli-input-json file://hello-world-deployment.json
```

El comando devuelve una respuesta similar al siguiente ejemplo.

```
{
  "deploymentId": "deb69c37-314a-4369-a6a1-3dff9fce73a9",
  "iotJobId": "b5d92151-6348-4941-8603-bdbfb3e02b75",
  "iotJobArn": "arn:aws:iot:region:account-id:job/b5d92151-6348-4941-8603-
bdbfb3e02b75"
}
```

3. Compruebe que la implementación se complete correctamente. La implementación puede tardar varios minutos en completarse. Consulte el registro de Hello World para comprobar el cambio. Ejecute el siguiente comando en su dispositivo principal de Greengrass.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
   ```

Debería ver mensajes similares al del siguiente ejemplo.

```
Hello, universe! Greetings from your first Greengrass component.
```

> **Note**
>
> Si los mensajes de registro no cambian, la implementación ha fallado o no ha llegado al dispositivo principal. Esto puede ocurrir si el dispositivo principal no está conectado a Internet o no tiene permisos para recuperar artefactos del bucket de S3. Ejecute el siguiente comando en su dispositivo principal para ver el archivo de registro del software AWS IoT Greengrass Core. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```
>
> Para obtener más información, consulte Solución de problemas AWS IoT Greengrass V2.

Pasos a seguir a continuación

Completó este tutorial. El software AWS IoT Greengrass Core y el componente Hello World se ejecutan en el dispositivo. Además, su componente Hello World está disponible en el servicio

en la nube de AWS IoT Greengrass para implementarlo en otros dispositivos. Para obtener más información sobre lo que se incluye en este tutorial, consulte lo siguiente:

- Creación de componentes de AWS IoT Greengrass
- Publique componentes para desplegarlos en sus dispositivos principales
- Implemente AWS IoT Greengrass componentes en los dispositivos

Configuración de los dispositivos AWS IoT Greengrass principales

Complete las tareas de esta sección para instalar, configurar y ejecutar el software AWS IoT Greengrass principal.

> **Note**
>
> En esta sección se describe la instalación y configuración avanzadas del software AWS IoT Greengrass principal. Estos pasos no se aplican a Nucleus Lite. Si es la primera vez que lo utiliza AWS IoT Greengrass V2, le recomendamos que complete primero el tutorial de introducción a la configuración de un dispositivo principal y explore sus funciones. AWS IoT Greengrass

Temas

- Plataformas admitidas
- Requisitos de los dispositivos
- Requisitos de la función de Lambda
- Configura un Cuenta de AWS
- Instalación del software AWS IoT Greengrass Core
- Ejecute el software AWS IoT Greengrass principal
- Ejecute AWS IoT Greengrass el software principal en un contenedor de Docker
- Configurar el software AWS IoT Greengrass principal
- Actualización del software AWS IoT Greengrass Core (OTA)
- Desinstalación del software AWS IoT Greengrass Core

Plataformas admitidas

- Plataformas compatibles con el núcleo de Greengrass
- Plataformas compatibles con la versión lite del núcleo de Greengrass

Requisitos de los dispositivos

- Requisitos del dispositivo del núcleo de Greengrass
- Requisitos del dispositivo de la versión lite del núcleo de Greengrass

Requisitos de la función de Lambda

> **Important**
>
> Actualmente, las funciones de Lambda de Greengrass no son compatibles con la versión lite del núcleo de Greengrass.

El dispositivo debe cumplir los siguientes requisitos para ejecutar las funciones de Lambda:

- Sistema operativo basado en Linux.
- El dispositivo debe tener el intérprete de comandos `mkfifo`.
- El dispositivo debe ejecutar las bibliotecas de lenguajes de programación que requiere una función de Lambda. Las bibliotecas necesarias deben instalarse en el dispositivo y agregarse a la variable de entorno `PATH`. Greengrass es compatible con todas las versiones compatibles con Lambda de los tiempos de ejecución de Python, Node.js y Java. Greengrass no aplica ninguna restricción adicional a las versiones de tiempo de ejecución de Lambda obsoletas. Para obtener más información sobre la AWS IoT Greengrass compatibilidad con los tiempos de ejecución de Lambda, consulte. Ejecución de funciones de AWS Lambda
- Para ejecutar funciones de Lambda en contenedores, su dispositivo debe cumplir los siguientes requisitos:
  - Kernel de Linux versión 4.4 o posterior.
  - El núcleo debe ser compatible con cgroups v1 y usted debe habilitar y montar los siguientes cgroups:
    - El grupo de memoria AWS IoT Greengrass para establecer el límite de memoria para las funciones Lambda en contenedores.
    - El cgroup de dispositivos para funciones de Lambda en contenedores con el fin de acceder a los dispositivos o volúmenes del sistema.

    El software AWS IoT Greengrass Core no es compatible con cgroups v2.

Para cumplir con este requisito, arranque el dispositivo con los siguientes parámetros del núcleo de Linux.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

> **Tip**
>
> En una Raspberry Pi, edite el archivo `/boot/cmdline.txt` para configurar los parámetros del núcleo del dispositivo.

- Las siguientes configuraciones del núcleo de Linux deben estar habilitadas en el dispositivo:
  - Espacio de nombres:
    - `CONFIG_IPC_NS`
    - `CONFIG_UTS_NS`
    - `CONFIG_USER_NS`
    - `CONFIG_PID_NS`
  - Grupos de control:
    - `CONFIG_CGROUP_DEVICE`
    - `CONFIG_CGROUPS`
    - `CONFIG_MEMCG`
  - Otros:
    - `CONFIG_POSIX_MQUEUE`
    - `CONFIG_OVERLAY_FS`
    - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`
    - `CONFIG_SECCOMP_FILTER`
    - `CONFIG_KEYS`
    - `CONFIG_SECCOMP`
    - `CONFIG_SHMEM`

> **Tip**
>
> Consulte la documentación de la distribución de Linux para saber cómo verificar y configurar los parámetros del núcleo de Linux. También puede utilizar AWS IoT Device

Tester for AWS IoT Greengrass para comprobar que el dispositivo cumple estos requisitos. Para obtener más información, consulte Uso de AWS IoT Device Tester para la versión 2 de AWS IoT Greengrass.

Configura un Cuenta de AWS

Si no tiene uno Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abrir https://portal.aws.amazon.com/billing/registro.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

   Cuando te registras en un Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar tareas que requieren acceso de usuario raíz.

Para crear un usuario administrador, elija una de las siguientes opciones.

Elegir una forma de administrar el administrador	Para	Haga esto	También puede
En IAM Identity Center	Usar credenciales a corto plazo para acceder a AWS.	Siga las instrucciones en Introducción en la Guía del usuario de AWS IAM Identity Center .	Configure el acceso programático configurando el AWS CLI que se utilizará AWS IAM Identity Center en la

Elegir una forma de administrar el administrador	Para	Haga esto	También puede
(recomendado)	Esto se ajusta a las prácticas recomendadas de seguridad. Para obtener información sobre las prácticas recomendadas, consulta Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.		Guía del AWS Command Line Interface usuario.
En IAM (no recomendado)	Usar credenciales a largo plazo para acceder a AWS.	Siguiendo las instrucciones de Crear un usuario de IAM para acceso de emergencia de la Guía del usuario de IAM.	Configure el acceso programático mediante Administrar las claves de acceso de los usuarios de IAM en la Guía del usuario de IAM.

Instalación del software AWS IoT Greengrass Core

AWS IoT Greengrass lleva AWS a los dispositivos de periferia, de modo que puedan actuar en función de los datos que generan, al mismo tiempo que utilizan la Nube de AWS para tareas de administración, análisis y almacenamiento duradero. Instale el software AWS IoT Greengrass Core en los dispositivos de periferia para integrarlo con AWS IoT Greengrass y la Nube de AWS.

> **Important**
>
> Antes de descargar e instalar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core versión 2.0.

El software AWS IoT Greengrass Core incluye un instalador que configura su dispositivo como un dispositivo principal de Greengrass. Al ejecutar el instalador, puede configurar las opciones, como la carpeta raíz y la carpeta Región de AWS a utilizar. Puede elegir que el instalador cree los recursos AWS IoT y de IAM necesarios para su caso. También puede optar por implementar herramientas de desarrollo local para configurar un dispositivo que utilice para el desarrollo de componentes personalizados.

El software AWS IoT Greengrass Core requiere lo siguiente AWS IoT y los recursos de IAM para conectarse a la Nube de AWS y funcionar:

- Un objeto de AWS IoT. Cuando registre un dispositivo como un objeto AWS IoT, ese dispositivo puede usar un certificado digital para autenticarse con AWS. Este certificado permite que el dispositivo se comunique con AWS IoT y AWS IoT Greengrass. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.
- (Opcional) Un grupo de objetos AWS IoT. Los grupos de objetos se usan para administrar las flotas de dispositivos principales de Greengrass. Al implementar componentes de software en sus dispositivos, puede optar por implementarlos en dispositivos individuales o en grupos de dispositivos. Puede agregar un dispositivo a un grupo de objetos para implementar los componentes de software de ese grupo de objetos en el dispositivo. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.
- Un rol de IAM. Los dispositivos principales de Greengrass utilizan el proveedor de credenciales de AWS IoT Core para autorizar las llamadas a los servicios de AWS con un rol de IAM. Este rol le permite a su dispositivo interactuar con AWS IoT, enviar registros a los Registros de Amazon CloudWatch y descargar artefactos de componentes personalizados de Amazon Simple Storage Service (Amazon S3). Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.
- Un alias de rol de AWS IoT. Los dispositivos principales de Greengrass utilizan el alias del rol para identificar el rol de IAM que se va a utilizar. El alias del rol le permite cambiar el rol de IAM, pero mantener la misma configuración del dispositivo. Para obtener más información, consulte

Autorización de llamadas a los servicios de AWS en la Guía para desarrolladores de AWS IoT Core.

Elija una de las siguientes opciones para instalar el software AWS IoT Greengrass Core en un dispositivo principal.

- Instalación rápida

  Elija esta opción para configurar un dispositivo principal de Greengrass en el menor número de pasos posible. El instalador crea los recursos AWS IoT de IAM necesarios para usted. Esta opción requiere que proporcione credenciales de AWS al instalador para crear recursos en su Cuenta de AWS.

  No puede usar esta opción para realizar la instalación detrás de un firewall o un proxy de red. Si sus dispositivos están protegidos por un firewall o un proxy de red, considere instalarlos manualmente.

  Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento automático de recursos.

- Instalación manual

  Elija esta opción para crear los recursos de AWS necesarios manualmente o para instalarlos detrás de un firewall o un proxy de red. Al realizar una instalación manual, no es necesario dar permiso al instalador para crear recursos en su Cuenta de AWS, ya que crea los recursos de AWS IoT y de IAM necesarios. También puede configurar el dispositivo para que se conecte al puerto 443 o a través de un proxy de red. También puede configurar el software AWS IoT Greengrass Core para que utilice una clave privada y un certificado que se almacenen en un módulo de seguridad de hardware (HSM), un módulo de plataforma segura (TPM) u otro elemento criptográfico.

  Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.

- Instalación con aprovisionamiento de flota AWS IoT

  Elija esta opción para crear los recursos de AWS necesarios a partir de una plantilla de aprovisionamiento de flotas AWS IoT. Puede elegir esta opción para crear dispositivos similares en una flota o si fabrica dispositivos que sus clientes activen más adelante, como vehículos o dispositivos domésticos inteligentes. Los dispositivos utilizan certificados identificativos para

autenticar y aprovisionar recursos de AWS, incluido un certificado de cliente X.509 que el dispositivo utiliza la Nube de AWS para conectarse al sistema operativo normal. Puede incrustar o archivar los certificados de reclamación en el equipo del dispositivo durante la fabricación y puede utilizar el mismo certificado de reclamación y la misma clave para aprovisionar varios dispositivos. También puede configurar dispositivos para que se conecte al puerto 443 o a través de un proxy de red.

Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota.

- Instalación con aprovisionamiento personalizado

  Elija esta opción para desarrollar una aplicación Java personalizada que aprovisione los recursos de AWS necesarios. Puede elegir esta opción si crea sus propios certificados de cliente X.509 o si desea tener más control sobre el proceso de aprovisionamiento. AWS IoT Greengrass proporciona una interfaz que puede implementar para intercambiar información entre su aplicación de aprovisionamiento personalizada y el instalador del software AWS IoT Greengrass Core.

  Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento de recursos personalizado.

AWS IoT Greengrass también proporciona entornos en contenedores que ejecutan el software de AWS IoT Greengrass Core. Puede usar un archivo Docker para ejecutar AWS IoT Greengrass en un contenedor de Docker.

Temas

- Instale el software AWS IoT Greengrass principal con aprovisionamiento automático de recursos
- Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos
- Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota
- Instale el software AWS IoT Greengrass principal con aprovisionamiento de recursos personalizado
- Argumentos del instalador

Instale el software AWS IoT Greengrass principal con aprovisionamiento automático de recursos

El software AWS IoT Greengrass Core incluye un instalador que configura su dispositivo como un dispositivo principal de Greengrass. Para configurar un dispositivo rápidamente, el instalador puede proporcionar la AWS IoT AWS IoT cosa, el grupo de cosas, la función de IAM y el alias de la AWS IoT función que el dispositivo principal necesita para funcionar. El instalador también puede implementar las herramientas de desarrollo locales en el dispositivo principal, de modo que usted pueda usar el dispositivo para desarrollar y probar componentes de software personalizados. El instalador necesita AWS credenciales para aprovisionar estos recursos y crear la implementación.

Si no puede proporcionar AWS las credenciales al dispositivo, puede aprovisionar los AWS recursos que el dispositivo principal necesita para funcionar. También puede implementar las herramientas de desarrollo en un dispositivo principal para usarlas como dispositivo de desarrollo. Esto le permite conceder menos permisos al dispositivo al ejecutar el instalador. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Configuración del entorno del dispositivo
- Proporcione AWS las credenciales al dispositivo
- Descargue el software AWS IoT Greengrass principal
- Instale el software principal AWS IoT Greengrass

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

   ```
   java -version
   ```

   El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT

Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

   ```
   sudo visudo
   ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:
   a. Pulse la tecla Windows para abrir el menú de inicio.
   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.
   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.
   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.
   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.
   f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

      ```
      C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
      ```

   g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.
3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.
4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

> **Note**
>
> En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Proporcione AWS las credenciales al dispositivo

Proporcione AWS las credenciales del dispositivo para que el instalador pueda aprovisionar los AWS recursos necesarios. Para obtener más información sobre los permisos necesarios, consulte Política de IAM mínima para que el instalador aprovisione recursos.

Para proporcionar AWS credenciales al dispositivo

- Proporcione sus AWS credenciales al dispositivo para que el instalador pueda aprovisionar los recursos de IAM AWS IoT y los de su dispositivo principal. Para aumentar la seguridad, le recomendamos que obtenga credenciales temporales para un rol de IAM que permita únicamente los permisos mínimos necesarios para el aprovisionamiento. Para obtener más información, consulte Política de IAM mínima para que el instalador aprovisione recursos.

  > **Note**
  >
  > El instalador no guarda ni almacena sus credenciales.

  En el dispositivo, realice una de las siguientes acciones para recuperar las credenciales y ponerlas a disposición del instalador del software AWS IoT Greengrass principal:

  - (Recomendado) Utilice credenciales temporales de AWS IAM Identity Center

    a. Proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde IAM Identity Center. Para obtener más información, consulte la Actualización manual de credenciales en Obtener y actualizar las credenciales temporales en la Guía del usuario de IAM Identity Center.

b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciales de seguridad temporales de un rol de (IAM):

a. Proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde un rol de IAM que asuma. Para obtener más información acerca de cómo recuperar estas credenciales, consulte Solicitud de credenciales de seguridad temporales en la Guía del usuario de IAM.

b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

```
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciales a largo plazo de un usuario de IAM:

  a. Proporcione el ID de clave de acceso y la clave de acceso secreta del usuario de IAM. Puede crear un usuario de IAM para el aprovisionamiento y luego eliminarlo. Para la política de IAM que debe proporcionarse al usuario, consulte Política de IAM mínima para que el instalador aprovisione recursos. Para obtener información acerca de cómo recuperar credenciales a largo plazo, consulte Administración de las claves de acceso de los usuarios de IAM en la Guía de usuario de IAM.

  b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass principal.

  Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

  Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

  PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

  c. (Opcional) Si creó un usuario de IAM para aprovisionar su dispositivo de Greengrass, elimínelo.

  d. (Opcional) Si utilizó el ID de clave de acceso y la clave de acceso secreta de un usuario de IAM existente, actualice las claves del usuario para que dejen de ser válidas. Para

obtener más información, consulte Actualización de claves de acceso en la Guía de usuario de AWS Identity and Access Management .

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

```
jar verified.
```

ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

```
jar is unsigned.
```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *`GreengrassInstaller`*Sustitúyalo por la carpeta que desee usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

> **Important**
>
> Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.

Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Instale el software principal AWS IoT Greengrass

Ejecute el instalador con argumentos que especifiquen lo siguiente:

- Cree los AWS recursos que el dispositivo principal necesita para funcionar.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.
- Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para configurar un dispositivo de desarrollo con herramientas de desarrollo local, especifique el argumento `--deploy-dev-tools true`. Una vez finalizada la instalación, la implementación de las herramientas de desarrollo local puede tardar hasta un minuto.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

Para instalar el software AWS IoT Greengrass Core

1. Ejecute el instalador AWS IoT Greengrass principal. Reemplace los valores de los argumentos en su comando de la siguiente manera.

   > **Note**
   >
   > Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

   a. */greengrass/v2*o bien*C:\greengrass\v2*: la ruta a la carpeta raíz que se utilizará para instalar el software AWS IoT Greengrass Core.

   b. *GreengrassInstaller*. La ruta a la carpeta en la que desempaquetó el instalador del software AWS IoT Greengrass Core.

   c. *region*. El Región de AWS lugar en el que encontrar o crear recursos.

   d. *MyGreengrassCore*. El nombre del AWS IoT dispositivo principal de Greengrass. Si el objeto no existe, el instalador la crea. El instalador descarga los certificados para autenticarse como tal. AWS IoT Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

      > **Note**
      >
      > El nombre del objeto no puede contener dos puntos (`:`).

   e. *MyGreengrassCoreGroup*. El nombre del grupo de AWS IoT cosas de su dispositivo principal de Greengrass. Si el grupo de objetos no existe, el instalador lo crea y le agrega un objeto. Si el grupo de objetos existe y tiene una implementación activa, el dispositivo principal descarga y ejecuta el software que especifique la implementación.

      > **Note**
      >
      > El nombre del grupo de objetos no puede contener dos puntos (`:`).

   f. *GreengrassV2IoTThingPolicy*. El nombre de la AWS IoT política que permite a los dispositivos principales de Greengrass comunicarse con AWS IoT y. AWS IoT Greengrass

Si la AWS IoT política no existe, el instalador crea una AWS IoT política permisiva con este nombre. Puede restringir los permisos de esta política según su caso de uso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

g. *GreengrassV2TokenExchangeRole*. El nombre de la función de IAM que permite al dispositivo principal de Greengrass obtener AWS credenciales temporales. Si el rol no existe, el instalador lo crea y asocia una política denominada *GreengrassV2TokenExchangeRole*`Access`. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

h. *GreengrassCoreTokenExchangeRoleAlias*. El alias de la función de IAM que permite al dispositivo principal de Greengrass obtener credenciales temporales más adelante. Si el alias del rol no existe, el instalador lo crea y lo dirige al rol de IAM que especifique. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
```

```
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass Core como un servicio del sistema.

El instalador imprime los siguientes mensajes si la operación es exitosa:

- Si especifica `--provision`, el instalador imprime `Successfully configured Nucleus with provisioned resource details` si configuró los recursos correctamente.
- Si especifica `--deploy-dev-tools`, el instalador imprime `Configured Nucleus to deploy aws.greengrass.Cli component` si creó la implementación correctamente.
- Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio.
- Si no especifica `--setup-system-service true`, el instalador imprime `Launched Nucleus successfully` si se ejecutó correctamente y ejecutó el software.

2. Omita este paso si instaló la versión 2.0.4 o una versión posterior de Núcleo de Greengrass. Si descargó la versión más reciente del software, instaló la versión 2.0.4 o una versión posterior.

Ejecute el siguiente comando para establecer los permisos de archivo necesarios para la carpeta raíz del software AWS IoT Greengrass principal. */greengrass/v2*Sustitúyala por la carpeta raíz que especificó en el comando de instalación y */greengrass* sustitúyala por la carpeta principal de la carpeta raíz.

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

Si instaló el software AWS IoT Greengrass principal como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

> **Note**
>
> De forma predeterminada, el rol de IAM que crea el instalador no permite el acceso a los artefactos de componentes de los buckets de S3. Para implementar componentes personalizados que definan artefactos en Amazon S3, debe agregar permisos al rol para permitir que su dispositivo principal recupere artefactos de componentes. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.
> Si aún no tiene un bucket de S3 para los artefactos de los componentes, puede agregar estos permisos más adelante, después de crear un bucket.

> **Note**
>
> Cuando el software AWS IoT Greengrass Core se conecte a la nube, su dispositivo será reconocido como un dispositivo Core.

Para obtener más información sobre cómo configurar y usar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos

El software AWS IoT Greengrass Core incluye un instalador que configura su dispositivo como un dispositivo principal de Greengrass. Para configurar un dispositivo manualmente, puede crear los recursos de IAM AWS IoT y los necesarios para que los utilice el dispositivo. Si crea estos recursos manualmente, no necesita proporcionar AWS las credenciales al instalador.

Al instalar manualmente el software AWS IoT Greengrass Core, también puede configurar el dispositivo para que utilice un proxy de red o se conecte AWS al puerto 443. Es posible que tenga que especificar estas opciones de configuración si su dispositivo funciona con un firewall o un proxy de red, por ejemplo. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

También puede configurar el software AWS IoT Greengrass Core para que utilice un módulo de seguridad de hardware (HSM) a través de la interfaz PKCS #11. Esta característica le permite almacenar de forma segura los archivos de certificados y claves privadas para que no queden expuestos ni duplicados en el software. Puede almacenar claves privadas y certificados en un módulo de hardware, como un HSM, un módulo de plataforma segura (TPM) u otro elemento criptográfico. Esta característica solo está disponible en dispositivos Linux. Para obtener más información sobre la seguridad del hardware y los requisitos para su uso, consulte Integración de la seguridad de hardware.

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Recupere los puntos finales AWS IoT
- Crea cualquier AWS IoT cosa
- Creación del certificado del objeto
- Configuración del certificado del objeto
- Creación de un rol de intercambio de token
- Descarga de certificados al dispositivo

- Configuración del entorno del dispositivo
- Descargue el software AWS IoT Greengrass principal
- Instale el software principal AWS IoT Greengrass

Recupere los puntos finales AWS IoT

Obtenga sus AWS IoT Cuenta de AWS puntos finales y guárdelos para usarlos más tarde. El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Haga lo siguiente:

1. Obtenga el punto final AWS IoT de datos para su. Cuenta de AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenga el punto final de AWS IoT credenciales para su Cuenta de AWS.

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

Crea cualquier AWS IoT cosa

AWS IoT las cosas representan dispositivos y entidades lógicas a las que se conectan AWS IoT. Los dispositivos principales de Greengrass son AWS IoT cosas. Cuando registras un dispositivo como una AWS IoT cosa, ese dispositivo puede usar un certificado digital para autenticarse. AWS

En esta sección, crearás AWS IoT algo que represente tu dispositivo.

Para crear cualquier AWS IoT cosa

1. Crea cualquier AWS IoT cosa para tu dispositivo. En su equipo de desarrollo, ejecute el siguiente comando.

   - *MyGreengrassCore*Sustitúyalo por el nombre de la cosa a utilizar. Este nombre también es el nombre de su dispositivo principal de Greengrass.

     > **Note**
     >
     > El nombre del objeto no puede contener dos puntos (`:`).

   ```
   aws iot create-thing --thing-name MyGreengrassCore
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "thingName": "MyGreengrassCore",
     "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
     "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
   }
   ```

2. (Opcional) Añada la AWS IoT cosa a un grupo de cosas nuevo o existente. Los grupos de objetos se usan para administrar las flotas de dispositivos principales de Greengrass. Al implementar componentes de software en sus dispositivos, puede dirigirlos a dispositivos individuales o a grupos de dispositivos. Puede agregar un dispositivo a un grupo de objetos con una implementación activa de Greengrass para implementar los componentes de software de ese grupo de objetos en el dispositivo. Haga lo siguiente:

   a. (Opcional) Cree un grupo de AWS IoT cosas.

      - *MyGreengrassCoreGroup*Sustitúyalo por el nombre del grupo de cosas que desee crear.

        > **Note**
        >
        > El nombre del grupo de objetos no puede contener dos puntos (`:`).

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. Añada la AWS IoT cosa a un grupo de cosas.

- *MyGreengrassCore*Sustitúyala por el nombre de la AWS IoT cosa.
- *MyGreengrassCoreGroup*Sustitúyalo por el nombre del grupo de cosas.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Creación del certificado del objeto

Al registrar un dispositivo como una AWS IoT cosa, ese dispositivo puede utilizar un certificado digital para autenticarse AWS. Este certificado permite que el dispositivo se comunique con AWS IoT y AWS IoT Greengrass.

En esta sección, puede crear y descargar certificados que el dispositivo puede usar para conectarse a AWS.

Si desea configurar el software AWS IoT Greengrass principal para que utilice un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado, siga los pasos para crear el certificado a partir de una clave privada de un HSM. De lo contrario, siga los pasos para crear el certificado y la clave privada en el AWS IoT servicio. La característica de seguridad de hardware solo está disponible en dispositivos Linux. Para obtener más información

sobre la seguridad del hardware y los requisitos para su uso, consulte Integración de la seguridad de hardware.

Cree el certificado y la clave privada en el AWS IoT servicio

Creación del certificado del objeto

1. Crea una carpeta donde descargues los certificados de la AWS IoT cosa.

```
mkdir greengrass-v2-certs
```

2. Crea y descarga los certificados de la AWS IoT cosa.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
```

```
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

Guarde el nombre de recurso de Amazon (ARN) del certificado para usarlo para configurar el certificado más adelante.

Creación del certificado a partir de una clave privada en un HSM

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Creación del certificado del objeto

1. En el dispositivo principal, inicialice un token PKCS#11 en el HSM y genere una clave privada. La clave privada debe ser una clave RSA con un tamaño de clave RSA-2048 (o mayor) o una clave ECC.

   > **Note**
   >
   > Para utilizar un módulo de seguridad de hardware con claves ECC, debe utilizar la versión del núcleo de Greengrass 2.5.6 o posterior.

> Para usar un módulo de seguridad de hardware y el administrador de secretos, debe usar un módulo de seguridad de hardware con claves RSA.

Consulte la documentación de su HSM para obtener información sobre cómo inicializar el token y generar la clave privada. Si su HSM admite objetos IDs, especifique un ID de objeto al generar la clave privada. Guarde el ID de ranura, el PIN de usuario, la etiqueta del objeto y el ID del objeto (si su HSM utiliza alguno) que especifique al inicializar el token y generar la clave privada. Estos valores se utilizan más adelante cuando se importa el certificado de la cosa al HSM y se configura el software AWS IoT Greengrass principal.

2. Cree una solicitud de firma de certificado (CSR) a partir de la clave privada. AWS IoT usa esta CSR para crear un certificado específico para la clave privada que generaste en el HSM. Para obtener información sobre cómo crear una CSR de la clave privada, consulte la documentación de su HSM. La CSR es un archivo, como `iotdevicekey.csr`.
3. Copie la CSR del dispositivo a su computadora de desarrollo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir la CSR. *`device-ip-address`*Sustitúyalo por la dirección IP del dispositivo y *`~/iotdevicekey.csr`* sustitúyalo por la ruta al archivo CSR del dispositivo.

   ```
   scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
   ```

4. En tu ordenador de desarrollo, crea una carpeta en la que descargues el certificado del dispositivo AWS IoT .

   ```
   mkdir greengrass-v2-certs
   ```

5. Utilice el archivo CSR para crear y descargar el certificado del dispositivo en AWS IoT su ordenador de desarrollo.

   ```
   aws iot create-certificate-from-csr --set-as-active --certificate-signing-
   request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
   device.pem.crt
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
   ```

```
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Guarde el ARN del certificado para usarlo más adelante para configurarlo.

Configuración del certificado del objeto

Adjunte el certificado del AWS IoT objeto al elemento que creó anteriormente y añada una AWS IoT política al certificado para definir los AWS IoT permisos del dispositivo principal.

Configuración del certificado del objeto

1. Adjunte el certificado a la AWS IoT cosa.

   - Reemplácelo *`MyGreengrassCore`* con el nombre de su AWS IoT cosa.
   - Reemplace el nombre de recurso de Amazon (ARN) del certificado por el ARN del certificado que creó en el paso anterior.

```
aws iot attach-thing-principal --thing-name MyGreengrassCore
 --principal arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

2. Cree y adjunte una AWS IoT política que defina los AWS IoT permisos de su dispositivo principal de Greengrass. La siguiente política permite el acceso a todos los temas de MQTT y a las operaciones de Greengrass, de modo que su dispositivo funcione con aplicaciones personalizadas y con cambios futuros que requieran nuevas operaciones de Greengrass. Puede restringir esta política en función del caso de uso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

   Si ya ha configurado un dispositivo principal de Greengrass, puede adjuntar su AWS IoT política en lugar de crear una nueva.

   Haga lo siguiente:

   a. Cree un archivo que contenga el documento AWS IoT de política que requieren los dispositivos principales de Greengrass.

      Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

      ```
      nano greengrass-v2-iot-policy.json
      ```

      Copie el siguiente JSON en el archivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "iot:Publish",
              "iot:Subscribe",
              "iot:Receive",
              "iot:Connect",
              "greengrass:*"
      ```

```
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

b. Cree una AWS IoT política a partir del documento de política.

- *GreengrassV2IoTThingPolicy*Sustitúyala por el nombre de la política que se va a crear.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17       \\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
```

```
}
```

c. Adjunta la AWS IoT política al certificado de la AWS IoT cosa.

- *GreengrassV2IoTThingPolicy*Sustitúyala por el nombre de la política que se va a adjuntar.
- Reemplace el ARN de destino por el ARN del certificado de su objeto AWS IoT .

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Creación de un rol de intercambio de token

Los dispositivos principales de Greengrass utilizan una función de servicio de IAM, denominada función de intercambio de fichas, para autorizar las llamadas a los servicios. AWS El dispositivo utiliza el proveedor de AWS IoT credenciales para obtener AWS credenciales temporales para esta función, lo que permite al dispositivo interactuar con Amazon Logs AWS IoT, enviar registros a Amazon CloudWatch Logs y descargar artefactos de componentes personalizados de Amazon S3. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Se utiliza un alias de AWS IoT rol para configurar el rol de intercambio de fichas para los dispositivos principales de Greengrass. Los alias de rol le permiten cambiar el rol de intercambio de token de un dispositivo, pero mantener la configuración del dispositivo igual. Para obtener más información, consulte Autorización de llamadas a los servicios de AWS en la Guía para desarrolladores de AWS IoT Core .

En esta sección, creará un rol de IAM de intercambio de tokens y un alias de AWS IoT rol que apunte al rol. Si ya ha configurado un dispositivo principal de Greengrass, puede usar su rol de intercambio de token y su alias de rol en lugar de crear otros nuevos. A continuación, configure el objeto AWS IoT del dispositivo para que use ese rol y ese alias.

Creación de un rol de IAM de intercambio de token

1. Creación de un rol de IAM que su dispositivo puede usar como rol de intercambio de token. Haga lo siguiente:

   a. Creación de un archivo que contenga el documento de política de confianza que requiere el rol de intercambio de token.

      Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

      ```
      nano device-role-trust-policy.json
      ```

      Copie el siguiente JSON en el archivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. Creación del rol de intercambio de token con el documento de política de confianza.

      - *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre del rol de IAM que se va a crear.

      ```
      aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
      policy-document file://device-role-trust-policy.json
      ```

      Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

      ```
      {
        "Role": {
      ```

```
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Creación de un archivo que contenga el documento de política de acceso que requiere el rol de intercambio de token.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano device-role-access-policy.json
   ```

   Copie el siguiente JSON en el archivo.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "logs:CreateLogGroup",
           "logs:CreateLogStream",
           "logs:PutLogEvents",
           "logs:DescribeLogStreams",
           "s3:GetBucketLocation"
         ],
         "Resource": "*"
   ```

```
    }
  ]
}
```

> **Note**
>
> Esta política de acceso no permite el acceso a los artefactos de componentes en los buckets de S3. Para implementar componentes personalizados que definan artefactos en Amazon S3, debe agregar permisos al rol para permitir que su dispositivo principal recupere artefactos de componentes. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.
>
> Si aún no tiene un bucket de S3 para los artefactos de los componentes, puede agregar estos permisos más adelante, después de crear un bucket.

d. Creación de la política de IAM a partir del documento de política.

- *GreengrassV2TokenExchangeRoleAccess*Sustitúyalo por el nombre de la política de IAM que se va a crear.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
```

```
}
```

e. Adjunte la política de IAM al rol de intercambio de token.

- Reemplace *GreengrassV2TokenExchangeRole* por el nombre del rol de IAM.
- Reemplace el ARN de la política por el ARN de la política de IAM que creó en el paso anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

2. Cree un alias de AWS IoT rol que apunte al rol de intercambio de fichas.

- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias del rol que se va a crear.
- Reemplace el ARN del rol por el ARN del rol de IAM que creó en el paso anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para crear un alias de rol, debe tener el permiso para transferir el rol de IAM de intercambio de token a AWS IoT. Si recibe un mensaje de error al intentar crear un alias de rol, compruebe que el AWS usuario tiene este permiso. Para obtener más información, consulte Conceder permisos a un usuario para transferir un rol a un AWS servicio en la Guía del AWS Identity and Access Management usuario.

3. Cree y adjunte una AWS IoT política que permita a su dispositivo principal de Greengrass utilizar el alias del rol para asumir el rol de intercambio de fichas. Si ya ha configurado un dispositivo principal de Greengrass, puede adjuntar su AWS IoT política de alias de rol en lugar de crear una nueva. Haga lo siguiente:

   a. (Opcional) Cree un archivo que contenga el documento AWS IoT de política que requiere el alias del rol.

      Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      Copie el siguiente JSON en el archivo.

      - Reemplace el ARN del recurso por el ARN del alias de rol.

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
      GreengrassCoreTokenExchangeRoleAlias"
          }
        ]
      }
      ```

   b. Cree una AWS IoT política a partir del documento de política.

      - *GreengrassCoreTokenExchangeRoleAliasPolicy*Sustitúyala por el nombre de la AWS IoT política que se va a crear.

      ```
      aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
       --policy-document file://greengrass-v2-iot-role-alias-policy.json
      ```

      Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17       \\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Adjunta la AWS IoT política al certificado de la AWS IoT cosa.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Sustitúyala por el nombre de la AWS IoT política de alias del rol.
- Reemplace el ARN de destino por el ARN del certificado de su objeto AWS IoT .

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Descarga de certificados al dispositivo

Anteriormente, descargó el certificado de su dispositivo en su computadora de desarrollo. En esta sección, copie el certificado en el dispositivo principal para configurar el dispositivo con los certificados que usa para conectarse a AWS IoT. También descargue el certificado de la autoridad del certificado raíz (CA) de Amazon. Si usa un HSM, también importe el archivo de certificado al HSM en esta sección.

- Si anteriormente creó el elemento certificado y clave privada en el AWS IoT servicio, siga los pasos para descargar los certificados con la clave privada y los archivos de certificado.
- Si anteriormente creó el certificado del objeto de una clave privada en un módulo de seguridad de hardware (HSM), siga los pasos para descargar los certificados con la clave privada y el certificado en un HSM.

Descargar certificados con clave privada y archivos de certificado

Descarga de certificados al dispositivo

1. Copia el AWS IoT certificado del objeto desde tu ordenador de desarrollo al dispositivo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir el certificado. *`device-ip-address`*Sustitúyalo por la dirección IP de tu dispositivo.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Cree la carpeta raíz de Greengrass en el dispositivo. Más adelante instalarás el software AWS IoT Greengrass principal en esta carpeta.

   > **Note**
   >
   > Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta que desee utilizar.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

```
mkdir C:\greengrass\v2
```

PowerShell

- Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

```
mkdir C:\greengrass\v2
```

3. (Solo para Linux) Establezca los permisos de la carpeta principal de la carpeta raíz de Greengrass.

   - */greengrass*Sustitúyalo por el elemento principal de la carpeta raíz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Copia los certificados de la AWS IoT cosa a la carpeta raíz de Greengrass.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta raíz de Greengrass.

   ```
   sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
   ```

5. Descarga el certificado de la autoridad de certificación raíz (CA) de Amazon. AWS IoT Los certificados están asociados al certificado de CA raíz de Amazon de forma predeterminada.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

Descarga de certificados con la clave privada y el certificado en un HSM

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Descarga de certificados al dispositivo

1. Copia AWS IoT el certificado de la máquina de desarrollo al dispositivo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir el certificado. *`device-ip-address`*Sustitúyalo por la dirección IP de tu dispositivo.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Cree la carpeta raíz de Greengrass en el dispositivo. Más adelante instalarás el software AWS IoT Greengrass principal en esta carpeta.

> **Note**
>
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

Linux or Unix

- Reemplace *`/greengrass/v2`* por la carpeta que desee utilizar.

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- Reemplace *`C:\greengrass\v2`* por la carpeta que desee utilizar.

```
mkdir C:\greengrass\v2
```

PowerShell

- Reemplace *`C:\greengrass\v2`* por la carpeta que desee utilizar.

```
mkdir C:\greengrass\v2
```

3. (Solo para Linux) Establezca los permisos de la carpeta principal de la carpeta raíz de Greengrass.

   - *`/greengrass`*Sustitúyalo por el elemento principal de la carpeta raíz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Importe el archivo de certificado del objeto `~/greengrass-v2-certs/device.pem.crt`, al HSM. Consulte la documentación de su HSM para saber cómo importar certificados allí. Importe

el certificado con el mismo token, ID de ranura, PIN de usuario, etiqueta de objeto e ID de objeto (si su HSM usa alguno) con los que generó la clave privada en el HSM anteriormente.

> **Note**
>
> Si generó la clave privada anteriormente sin un ID de objeto y el certificado tiene un ID de objeto, establezca el ID de objeto de la clave privada con el mismo valor que el certificado. Consulte la documentación de su HSM para obtener información sobre cómo configurar el identificador de objeto para el objeto de clave privada.

5. (Opcional) Elimine el archivo de certificado del objeto para que solo exista en el HSM.

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. Descarga el certificado de la autoridad de certificación raíz (CA) de Amazon. AWS IoT Los certificados están asociados al certificado de CA raíz de Amazon de forma predeterminada.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

   ```
   java -version
   ```

   El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT

Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

   ```
   sudo visudo
   ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:
   a. Pulse la tecla Windows para abrir el menú de inicio.
   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.
   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.
   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.
   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.
   f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

      ```
      C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
      ```

   g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.
3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.
4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

Tip

Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.

- Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.

  ```
  net user ggc_user | findstr /C:expires
  ```

- Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.

  ```
  wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
  ```

- Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.

  ```
  Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
   'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
  ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

> **Note**
>
> En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

   i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

   ```
   jar verified.
   ```

   ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

   ```
   jar is unsigned.
   ```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *`GreengrassInstaller`*Sustitúyalo por la carpeta que desee usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.
> Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Instale el software principal AWS IoT Greengrass

Ejecute el instalador con argumentos que especifiquen las siguientes acciones:

- Instálelo desde un archivo de configuración parcial que especifique el uso de AWS los recursos y certificados que creó anteriormente. El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica la configuración de todos los componentes de Greengrass del dispositivo. El instalador crea un archivo de configuración completo a partir del archivo de configuración parcial que usted proporciona.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.
- Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

- Si creó anteriormente el certificado y la clave privada en el AWS IoT servicio, siga los pasos para instalar el software AWS IoT Greengrass Core con los archivos de clave privada y certificado.
- Si anteriormente creó el certificado Thing a partir de una clave privada en un módulo de seguridad de hardware (HSM), siga los pasos para instalar el software AWS IoT Greengrass Core con la clave privada y el certificado en un HSM.

Instale el software AWS IoT Greengrass principal con la clave privada y los archivos de certificado

Para instalar el software AWS IoT Greengrass Core

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *GreengrassInstaller*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema y los parámetros del núcleo de Greengrass.

   ```
   ---
   ```

```
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
```

A continuación, proceda del modo siguiente:

- Reemplace cada instancia de */greengrass/v2* por la carpeta raíz de Greengrass.
- *MyGreengrassCore*Sustitúyalo por el nombre de la AWS IoT cosa.
- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- *us-west-2*Sustitúyala por la Región de AWS ubicación en la que creaste los recursos.
- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias de la función de intercambio de fichas.
- `iotDataEndpoint`Sustitúyalo por el punto final de AWS IoT datos.
- Sustituya el `iotCredEndpoint` punto final por el de sus AWS IoT credenciales.

> **Note**
>
> En este archivo de configuración, puede personalizar otras opciones de configuración del núcleo, como los puertos y el proxy de red que se utilizarán, tal como se muestra en el siguiente ejemplo. Para obtener más información, consulte la Configuración del núcleo de Greengrass.
>
> ```
> ---
> system:
>   certificateFilePath: "/greengrass/v2/device.pem.crt"
> ```

```
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. Ejecute el instalador y especifique `--init-config` para proporcionar el archivo de configuración.

   - Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.
   - Sustituya cada instancia de por *GreengrassInstaller* la carpeta en la que desempaquetó el instalador.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

> **Note**
>
> No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`. Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

4. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Instale el software AWS IoT Greengrass Core con la clave privada y el certificado en un HSM

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Para instalar el software AWS IoT Greengrass principal

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *GreengrassInstaller*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Para permitir que el software AWS IoT Greengrass principal utilice la clave privada y el certificado del HSM, instale el componente de proveedor PKCS #11 al instalar el software AWS IoT Greengrass principal. El componente de proveedor PKCS#11 es un complemento que puede

configurar durante la instalación. Puede descargar la versión más reciente del componente de proveedor PKCS#11 de la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-latest.jar

Descargue el complemento de proveedor PKCS#11 en un archivo denominado `aws.greengrass.crypto.Pkcs11Provider.jar`. Sustitúyala por la carpeta que quieras usar. *`GreengrassInstaller`*

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

3. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema, los parámetros del núcleo de Greengrass y los parámetros del proveedor PKCS#11.

   ```
   ---
   system:
     certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
     privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.16.1"
       configuration:
   ```

```
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

A continuación, proceda del modo siguiente:

- Sustituya cada instancia del PKCS #11 URIs por la etiqueta de objeto *iotdevicekey* en la que creó la clave privada e importó el certificado.
- Reemplace cada instancia de */greengrass/v2* por la carpeta raíz de Greengrass.
- *MyGreengrassCore*Sustitúyala por el nombre de la AWS IoT cosa.
- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- *us-west-2*Sustitúyala por la Región de AWS ubicación en la que creaste los recursos.
- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias de la función de intercambio de fichas.
- `iotDataEndpoint`Sustitúyalo por el punto final de AWS IoT datos.
- Sustituya el `iotCredEndpoint` punto final por el de sus AWS IoT credenciales.
- Reemplace los parámetros de configuración del componente `aws.greengrass.crypto.Pkcs11Provider` por los valores de la configuración del HSM en el dispositivo principal.

> **Note**
>
> En este archivo de configuración, puede personalizar otras opciones de configuración del núcleo, como los puertos y el proxy de red que se utilizarán, tal como se muestra en el siguiente ejemplo. Para obtener más información, consulte la Configuración del núcleo de Greengrass.
>
> ```
> ---
> system:
> ```

```
    certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
    privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
    rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
    rootpath: "/greengrass/v2"
    thingName: "MyGreengrassCore"
  services:
    aws.greengrass.Nucleus:
      componentType: "NUCLEUS"
      version: "2.16.1"
      configuration:
        awsRegion: "us-west-2"
        iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
        iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
        iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
  west-2.amazonaws.com"
        mqtt:
          port: 443
        greengrassDataPlanePort: 443
        networkProxy:
          noProxyAddresses: "http://192.168.0.1,www.example.com"
          proxy:
            url: "https://my-proxy-server:1100"
            username: "Mary_Major"
            password: "pass@word1357"
    aws.greengrass.crypto.Pkcs11Provider:
      configuration:
        name: "softhsm_pkcs11"
        library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
        slot: 1
        userPin: "1234"
```

4. Ejecute el instalador y especifique `--init-config` para proporcionar el archivo de configuración.

   - Reemplace */greengrass/v2* por la carpeta raíz de Greengrass.
   - Sustituya cada instancia de *GreengrassInstaller* por la carpeta en la que desempaquetó el instalador.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
```

```
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

> **Note**
>
> No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`. Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

5. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Si instaló el software AWS IoT Greengrass Core como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

> **Note**
>
> Cuando el software AWS IoT Greengrass Core se conecte a la nube, su dispositivo será reconocido como un dispositivo Core.

Para obtener más información sobre cómo configurar y usar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass.

Con el aprovisionamiento de AWS IoT flotas, puede configurarlo AWS IoT para generar y entregar de forma segura certificados de dispositivo X.509 y claves privadas a sus dispositivos cuando se conecten a ellos AWS IoT por primera vez. AWS IoT proporciona certificados de cliente firmados por la autoridad de certificación (CA) raíz de Amazon. También puede configurarlo AWS IoT para especificar grupos de cosas, tipos de cosas y permisos para los dispositivos principales de Greengrass que aprovisiona con el aprovisionamiento de flotas. Defina una plantilla de aprovisionamiento para definir cómo AWS IoT aprovisiona cada dispositivo. En la plantilla de aprovisionamiento se especifica el objeto, la política y los recursos de certificado que se van a crear

para un dispositivo durante el aprovisionamiento. Para más información, consulte Plantillas de aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

AWS IoT Greengrass proporciona un complemento de aprovisionamiento de AWS IoT flotas que puede usar para instalar el software AWS IoT Greengrass principal utilizando los AWS recursos creados por el aprovisionamiento de AWS IoT flotas. El complemento de aprovisionamiento de flotas utiliza el aprovisionamiento por reclamación. Los dispositivos utilizan un certificado de aprovisionamiento y una clave privada para obtener un certificado de dispositivo X.509 único y una clave privada que pueden utilizar para operaciones habituales. Puede incrustar el certificado de reclamación y la clave privada en cada dispositivo durante la fabricación, de modo que luego los clientes puedan activar los dispositivos, cuando cada dispositivo esté en funcionamiento. Puede utilizar el mismo certificado de reclamación y clave privada para varios dispositivos. Para obtener más información, consulte Aprovisionamiento por reclamación en la Guía para desarrolladores de AWS IoT Core .

> **Note**
>
> Actualmente, el complemento de aprovisionamiento de flota no admite el almacenamiento de archivos de certificados y claves privadas en un módulo de seguridad de hardware (HSM). Para usar un HSM, instale el software AWS IoT Greengrass Core con aprovisionamiento manual.

Para instalar el software AWS IoT Greengrass Core con el aprovisionamiento de AWS IoT flota, debe configurar los recursos Cuenta de AWS que AWS IoT utiliza para aprovisionar los dispositivos principales de Greengrass. Estos recursos incluyen una plantilla de aprovisionamiento, certificados de reclamación y un rol de IAM para el intercambio de token. Después de crear estos recursos, puede reutilizarlos para aprovisionar varios dispositivos principales de una flota. Para obtener más información, consulte Configurar el aprovisionamiento de AWS IoT flota para los dispositivos principales de Greengrass.

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Requisitos previos
- Recupere los puntos finales AWS IoT
- Descarga de certificados al dispositivo
- Configuración del entorno del dispositivo
- Descargue el software AWS IoT Greengrass principal
- Descargue el complemento de aprovisionamiento AWS IoT de flotas
- Instale el software AWS IoT Greengrass principal
- Configurar el aprovisionamiento de AWS IoT flota para los dispositivos principales de Greengrass
- Configurar el complemento de aprovisionamiento de AWS IoT flotas
- Registro de cambios del complemento de aprovisionamiento de flotas AWS IoT

Requisitos previos

Para instalar el software AWS IoT Greengrass Core con el aprovisionamiento de AWS IoT flota, primero debe configurar el aprovisionamiento de AWS IoT flota para los dispositivos principales de Greengrass. Tras completar estos pasos una vez, puede utilizar el aprovisionamiento de flotas para instalar el software AWS IoT Greengrass Core en cualquier número de dispositivos.

Recupere los puntos finales AWS IoT

Obtenga los AWS IoT puntos finales que desee y guárdelos para usarlos más adelante. Cuenta de AWS El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Haga lo siguiente:

1. Obtenga el punto final AWS IoT de datos para su. Cuenta de AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenga el punto final de AWS IoT credenciales para su Cuenta de AWS.

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Descarga de certificados al dispositivo

El dispositivo utiliza un certificado de reclamación y una clave privada para autenticar su solicitud de aprovisionamiento de AWS recursos y adquirir un certificado de dispositivo X.509. Puede incrustar el certificado de reclamación y la clave privada en el dispositivo durante la fabricación o copiar el certificado y la clave en el dispositivo durante la instalación. En esta sección, debe copiar el certificado de reclamación y la clave privada en el dispositivo. También descargue el certificado de la autoridad del certificado raíz (CA) de Amazon en el dispositivo.

> **Important**
>
> Las claves privadas de la notificación de aprovisionamiento deben estar protegidas en todo momento, también en el dispositivo principal de Greengrass. Te recomendamos que utilices CloudWatch las estadísticas y los registros de Amazon para detectar indicios de uso indebido, como el uso no autorizado del certificado de reclamación para aprovisionar dispositivos. Si detecta un uso incorrecto, deshabilite el certificado de notificación de aprovisionamiento para que no se pueda utilizar para el aprovisionamiento de dispositivos. Para obtener más información, consulte Monitorear AWS IoT en la Guía para desarrolladores de AWS IoT Core .
>
> Para ayudarte a gestionar mejor el número de dispositivos y los dispositivos que se registran automáticamente en el tuyo Cuenta de AWS, puedes especificar un enlace previo al aprovisionamiento al crear una plantilla de aprovisionamiento de flotas. Un enlace de preaprovisionamiento es una AWS Lambda función que valida los parámetros de plantilla que proporcionan los dispositivos durante el registro. Por ejemplo, puede crear un enlace previo al aprovisionamiento que compruebe un ID de un dispositivo con una base de datos para comprobar que el dispositivo tiene permiso de aprovisionamiento. Para obtener

más información, consulte los enlaces previos al aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

Cómo descargar los certificados de reclamación al dispositivo

1. Copie el certificado de reclamación y la clave privada en el dispositivo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir el certificado de reclamación y la clave privada. El siguiente comando de ejemplo transfiere estos archivos a una carpeta denominada `claim-certs` en la computadora de desarrollo al dispositivo. *`device-ip-address`*Sustitúyala por la dirección IP de tu dispositivo.

   ```
   scp -r claim-certs/ device-ip-address:~
   ```

2. Cree la carpeta raíz de Greengrass en el dispositivo. Más adelante instalarás el software AWS IoT Greengrass principal en esta carpeta.

   > **Note**
   >
   > Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta que desee utilizar.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   mkdir C:\greengrass\v2
   ```

PowerShell

- Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

```
mkdir C:\greengrass\v2
```

3. (Solo para Linux) Establezca los permisos de la carpeta principal de la carpeta raíz de Greengrass.

   - */greengrass*Sustitúyalo por el elemento principal de la carpeta raíz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Mueva los certificados de reclamación a la carpeta raíz de Greengrass.

   - Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.

   Linux or Unix

   ```
   sudo mv ~/claim-certs /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   move %USERPROFILE%\claim-certs C:\greengrass\v2
   ```

   PowerShell

   ```
   mv -Path ~\claim-certs -Destination C:\greengrass\v2
   ```

5. Descarga el certificado de la autoridad de certificación raíz (CA) de Amazon. AWS IoT Los certificados están asociados al certificado de CA raíz de Amazon de forma predeterminada.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

```
java -version
```

El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

   ```
   sudo visudo
   ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:

   a. Pulse la tecla Windows para abrir el menú de inicio.

   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.

   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.

   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.

   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.

f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.

3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.

4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el `wmic`comando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.

6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

   > **Note**
   >
   > En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

```
jar verified.
```

ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

```
jar is unsigned.
```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *GreengrassInstaller*Sustitúyalo por la carpeta que desee usar.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
```

```
rm greengrass-nucleus-latest.zip
```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.
> Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Descargue el complemento de aprovisionamiento AWS IoT de flotas

Puedes descargar la última versión del complemento de aprovisionamiento de AWS IoT flotas desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar

> **Note**
>
> Puede descargar una versión específica del complemento de aprovisionamiento de AWS IoT flotas desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar. Para obtener más información sobre cada versión del complemento de aprovisionamiento de flota, consulte Registro de cambios del complemento de aprovisionamiento de flotas AWS IoT.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
> FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
> ```

El complemento de aprovisionamiento de flota es de código abierto. Para ver su código fuente, consulta el complemento de aprovisionamiento de AWS IoT flotas en GitHub.

Para descargar el complemento de aprovisionamiento de AWS IoT flotas

- En su dispositivo, descargue el complemento de aprovisionamiento de AWS IoT flotas en un archivo denominado. `aws.greengrass.FleetProvisioningByClaim.jar` *`GreengrassInstaller`*Sustitúyalo por la carpeta que desee usar.

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  PowerShell

  ```
  iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -
  OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

Instale el software AWS IoT Greengrass principal

Ejecute el instalador con argumentos que especifiquen las siguientes acciones:

- Instálelo desde un archivo de configuración parcial que especifique el uso del complemento de aprovisionamiento de flotas para aprovisionar AWS recursos. El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica la configuración de todos los componentes de Greengrass del dispositivo. El instalador crea un archivo de configuración completo a partir

del archivo de configuración parcial que usted proporciona y de los recursos de AWS que crea el complemento de aprovisionamiento de flota.

- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.
- Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

Para instalar el software AWS IoT Greengrass Core

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *`GreengrassInstaller`*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano GreengrassInstaller/config.yaml
```

Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del complemento de aprovisionamiento de flota. Para obtener más información acerca de las opciones que puede especificar, consulte Configurar el complemento de aprovisionamiento de AWS IoT flotas.

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
```

```
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

A continuación, proceda del modo siguiente:

- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- Sustituya cada instancia de */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.

> **Note**
>
> En los dispositivos Windows, debe especificar los separadores de rutas como barras invertidas dobles (\\), como `C:\\greengrass\\v2`.

- *us-west-2*Sustitúyala por la AWS región en la que creaste la plantilla de aprovisionamiento y otros recursos.
- `iotDataEndpoint`Sustitúyalo por su punto final AWS IoT de datos.
- Sustituya el `iotCredentialEndpoint` punto final por el de sus AWS IoT credenciales.
- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias de la función de intercambio de fichas.
- *GreengrassFleetProvisioningTemplate*Sustitúyalo por el nombre de la plantilla de aprovisionamiento de flota.
- Sustituya `claimCertificatePath` por la ruta al certificado de reclamación del dispositivo.

- Sustituya `claimCertificatePrivateKeyPath` por la ruta a la clave privada del certificado de reclamación del dispositivo.
- Sustituya los parámetros de la plantilla (`templateParameters`) por los valores que se usan para aprovisionar el dispositivo. Este ejemplo hace referencia a la plantilla de ejemplo que define los parámetros `ThingName` y `ThingGroupName`.

> **Note**
>
> En este archivo de configuración, puede personalizar otras opciones de configuración, como los puertos y el proxy de red que utilice, tal como se muestra en el siguiente ejemplo. Para obtener más información, consulte la Configuración del núcleo de Greengrass.
>
> Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
```

```
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\
\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
```

```
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Para utilizar un proxy HTTPS, debe utilizar la versión 1.1.0 o posterior del complemento de aprovisionamiento de flotas. Además, debe especificar la `rootCaPath` de `system`, como se muestra en el siguiente ejemplo.

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. Ejecute el instalador. Especifique `--trusted-plugin` si desea proporcionar el complemento de aprovisionamiento de flota y especifique `--init-config` si desea proporcionar el archivo de configuración.

   - Reemplace */greengrass/v2* por la carpeta raíz de Greengrass.
   - Sustituya cada instancia de por *GreengrassInstaller* la carpeta en la que desempaquetó el instalador.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
   ```

```
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

> **Note**
>
> No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`. Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

4. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Si instaló el software AWS IoT Greengrass Core como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

Para obtener más información sobre cómo configurar y utilizar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Configurar el aprovisionamiento de AWS IoT flota para los dispositivos principales de Greengrass

Para instalar el software AWS IoT Greengrass Core con el aprovisionamiento de flotas, primero debe configurar los siguientes recursos en su. Cuenta de AWS Estos recursos permiten que los dispositivos se registren AWS IoT y funcionen como dispositivos principales de Greengrass. Siga los pasos de esta sección una vez para crear y configurar estos recursos en su Cuenta de AWS.

- Un rol de IAM de intercambio de token, que los dispositivos principales utilizan para autorizar las llamadas a los servicios de AWS .
- Un alias de AWS IoT rol que apunta al rol de intercambio de fichas.
- (Opcional) Una AWS IoT política que los dispositivos principales utilizan para autorizar las llamadas a los AWS IoT Greengrass servicios AWS IoT y. Esta AWS IoT política debe permitir el `iot:AssumeRoleWithCertificate` permiso para el alias de la AWS IoT función que apunta a la función de intercambio de fichas.

  Puede usar una AWS IoT política única para todos los dispositivos principales de su flota, o puede configurar la plantilla de aprovisionamiento de la flota para crear una AWS IoT política para cada dispositivo principal.
- Una plantilla de aprovisionamiento de AWS IoT flota. Esta plantilla debe especificar lo siguiente:
  - Cualquier AWS IoT cosa, un recurso. Puede especificar una lista de grupos de objetos existentes para implementar componentes en cada dispositivo cuando esté en línea.
  - Un recurso AWS IoT político. Este recurso puede definir al menos una de las siguientes propiedades:
    - El nombre de una AWS IoT política existente. Si elige esta opción, los dispositivos principales que cree a partir de esta plantilla utilizarán la misma AWS IoT política y podrá gestionar sus permisos como una flota.
    - Un documento AWS IoT de política. Si elige esta opción, cada dispositivo principal que cree a partir de esta plantilla utilizará una AWS IoT política única y podrá administrar los permisos de cada dispositivo principal individual.
  - Un recurso AWS IoT de certificados. Este recurso de certificado debe usar el parámetro `AWS::IoT::Certificate::Id` para adjuntar el certificado al dispositivo principal. Para obtener más información, consulte el Just-in-time aprovisionamiento en la Guía para AWS IoT desarrolladores.

- Un certificado de notificación de AWS IoT aprovisionamiento y una clave privada para la plantilla de aprovisionamiento de flotas. Puede incrustar este certificado y esta clave privada en los dispositivos durante la fabricación, de modo que los dispositivos puedan registrarse y aprovisionarse por sí mismos cuando se conecten a Internet.

  > **Important**
  >
  > Las claves privadas de la notificación de aprovisionamiento deben estar protegidas en todo momento, también en el dispositivo principal de Greengrass. Te recomendamos que utilices CloudWatch las estadísticas y los registros de Amazon para detectar indicios de uso indebido, como el uso no autorizado del certificado de reclamación para aprovisionar dispositivos. Si detecta un uso incorrecto, deshabilite el certificado de notificación de aprovisionamiento para que no se pueda utilizar para el aprovisionamiento de dispositivos. Para obtener más información, consulte Monitorear AWS IoT en la Guía para desarrolladores de AWS IoT Core .
  >
  > Para ayudarte a gestionar mejor el número de dispositivos y los dispositivos que se registran automáticamente en el tuyo Cuenta de AWS, puedes especificar un enlace previo al aprovisionamiento al crear una plantilla de aprovisionamiento de flotas. Un enlace de preaprovisionamiento es una AWS Lambda función que valida los parámetros de plantilla que proporcionan los dispositivos durante el registro. Por ejemplo, puede crear un enlace previo al aprovisionamiento que compruebe un ID de un dispositivo con una base de datos para comprobar que el dispositivo tiene permiso de aprovisionamiento. Para obtener más información, consulte los enlaces previos al aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

- AWS IoT Política que se adjunta al certificado de notificación de aprovisionamiento para permitir que los dispositivos se registren y utilicen la plantilla de aprovisionamiento de flota.

Temas

- Creación de un rol de intercambio de token
- Cree una AWS IoT política
- Creación de una plantilla de aprovisionamiento de flota
- Creación de un certificado de notificación de aprovisionamiento y una clave privada

Creación de un rol de intercambio de token

Los dispositivos principales de Greengrass utilizan una función de servicio de IAM, denominada función de intercambio de fichas, para autorizar las llamadas a los servicios. AWS El dispositivo utiliza el proveedor de AWS IoT credenciales para obtener AWS credenciales temporales para esta función, lo que permite al dispositivo interactuar con Amazon Logs AWS IoT, enviar registros a Amazon CloudWatch Logs y descargar artefactos de componentes personalizados de Amazon S3. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Se utiliza un alias de AWS IoT rol para configurar el rol de intercambio de fichas para los dispositivos principales de Greengrass. Los alias de rol le permiten cambiar el rol de intercambio de token de un dispositivo, pero mantener la configuración del dispositivo igual. Para obtener más información, consulte Autorización de llamadas a los servicios de AWS en la Guía para desarrolladores de AWS IoT Core .

En esta sección, creará un rol de IAM de intercambio de tokens y un alias de AWS IoT rol que apunte al rol. Si ya ha configurado un dispositivo principal de Greengrass, puede usar su rol de intercambio de token y su alias de rol en lugar de crear otros nuevos.

Creación de un rol de IAM de intercambio de token

1. Creación de un rol de IAM que su dispositivo puede usar como rol de intercambio de token. Haga lo siguiente:

   a. Creación de un archivo que contenga el documento de política de confianza que requiere el rol de intercambio de token.

      Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

      ```
      nano device-role-trust-policy.json
      ```

      Copie el siguiente JSON en el archivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
      ```

```
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Creación del rol de intercambio de token con el documento de política de confianza.

- *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre del rol de IAM que se va a crear.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Creación de un archivo que contenga el documento de política de acceso que requiere el rol de intercambio de token.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano device-role-access-policy.json
```

Copie el siguiente JSON en el archivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Esta política de acceso no permite el acceso a los artefactos de componentes en los buckets de S3. Para implementar componentes personalizados que definan artefactos en Amazon S3, debe agregar permisos al rol para permitir que su dispositivo principal recupere artefactos de componentes. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.
> Si aún no tiene un bucket de S3 para los artefactos de los componentes, puede agregar estos permisos más adelante, después de crear un bucket.

d. Creación de la política de IAM a partir del documento de política.

- *GreengrassV2TokenExchangeRoleAccess*Sustitúyalo por el nombre de la política de IAM que se va a crear.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Adjunte la política de IAM al rol de intercambio de token.

- Reemplace *GreengrassV2TokenExchangeRole* por el nombre del rol de IAM.
- Reemplace el ARN de la política por el ARN de la política de IAM que creó en el paso anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

2. Cree un alias de AWS IoT rol que apunte al rol de intercambio de fichas.

- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias del rol que se va a crear.
- Reemplace el ARN del rol por el ARN del rol de IAM que creó en el paso anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para crear un alias de rol, debe tener el permiso para transferir el rol de IAM de intercambio de token a AWS IoT. Si recibe un mensaje de error al intentar crear un alias de rol, compruebe que el AWS usuario tiene este permiso. Para obtener más información, consulte Conceder permisos a un usuario para transferir un rol a un AWS servicio en la Guía del AWS Identity and Access Management usuario.

Cree una AWS IoT política

Después de registrar un dispositivo como una AWS IoT cosa, ese dispositivo puede usar un certificado digital para autenticarse AWS. Este certificado incluye una o más AWS IoT políticas que definen los permisos que un dispositivo puede usar con el certificado. Estas políticas permiten que el dispositivo se comunique con AWS IoT y AWS IoT Greengrass.

Con el aprovisionamiento de AWS IoT flotas, los dispositivos se conectan AWS IoT para crear y descargar un certificado de dispositivo. En la plantilla de aprovisionamiento de flotas que cree en la siguiente sección, puede especificar si desea AWS IoT adjuntar la misma AWS IoT política a todos los certificados de los dispositivos o crear una nueva política para cada dispositivo.

En esta sección, creará una AWS IoT política que se AWS IoT adjunte a los certificados de todos los dispositivos. Con este enfoque, puede administrar los permisos de todos los dispositivos como una flota. Si prefieres crear una AWS IoT política nueva para cada dispositivo, puedes saltarte esta sección y consultar la política que contiene cuando definas la plantilla de tu flota.

Para crear una AWS IoT política

- Cree una AWS IoT política que defina los AWS IoT permisos para su flota de dispositivos principales de Greengrass. La siguiente política permite el acceso a todos los temas de MQTT y a las operaciones de Greengrass, de modo que su dispositivo funcione con aplicaciones personalizadas y con cambios futuros que requieran nuevas operaciones de Greengrass. Esta política también concede el permiso `iot:AssumeRoleWithCertificate`, que permite a los dispositivos utilizar el rol de intercambio de token que creó en la sección anterior. Puede restringir esta política en función del caso de uso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

  Haga lo siguiente:

  a. Cree un archivo que contenga el documento AWS IoT de política que requieren los dispositivos principales de Greengrass.

     Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

     ```
     nano greengrass-v2-iot-policy.json
     ```

     Copie el siguiente JSON en el archivo.

     - Sustituya el `iot:AssumeRoleWithCertificate` recurso por el ARN del alias de AWS IoT rol que creó en la sección anterior.

     JSON

     ```
     {
       "Version":"2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Action": [
             "iot:Publish",
             "iot:Subscribe",
             "iot:Receive",
             "iot:Connect",
             "greengrass:*"
     ```

```
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Cree una AWS IoT política a partir del documento de política.

- *GreengrassV2IoTThingPolicy*Sustitúyala por el nombre de la política que se va a crear.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

JSON

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
```

```
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

Creación de una plantilla de aprovisionamiento de flota

AWS IoT Las plantillas de aprovisionamiento de flotas definen cómo aprovisionar AWS IoT los artículos, las políticas y los certificados. Para aprovisionar los dispositivos principales de Greengrass con el complemento de aprovisionamiento de flotas, debe crear una plantilla en la que se especifique lo siguiente:

- Cualquier AWS IoT cosa, un recurso. Puede especificar una lista de grupos de objetos existentes para implementar componentes en cada dispositivo cuando esté en línea.
- Un recurso AWS IoT político. Este recurso puede definir al menos una de las siguientes propiedades:
  - El nombre de una AWS IoT política existente. Si elige esta opción, los dispositivos principales que cree a partir de esta plantilla utilizarán la misma AWS IoT política y podrá gestionar sus permisos como una flota.
  - Un documento AWS IoT de política. Si elige esta opción, cada dispositivo principal que cree a partir de esta plantilla utilizará una AWS IoT política única y podrá administrar los permisos de cada dispositivo principal individual.
- Un recurso AWS IoT de certificados. Este recurso de certificado debe usar el parámetro `AWS::IoT::Certificate::Id` para adjuntar el certificado al dispositivo principal. Para obtener más información, consulte el Just-in-time aprovisionamiento en la Guía para AWS IoT desarrolladores.

En la plantilla, puede especificar si desea añadirlo a una lista de grupos de cosas existentes. AWS IoT Cuando el dispositivo principal se conecta AWS IoT Greengrass por primera vez, recibe despliegues de Greengrass para cada grupo de elementos del que es miembro. Puede usar grupos de objetos para implementar el software más reciente en cada dispositivo tan pronto como esté en línea. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

El AWS IoT servicio requiere permisos para crear y actualizar sus AWS IoT recursos Cuenta de AWS al aprovisionar sus dispositivos. Para dar acceso al AWS IoT servicio, debe crear un rol de IAM y proporcionarlo al crear la plantilla. AWS IoT proporciona una política gestionada, el AWSIoTThingsregistro, que permite el acceso a todos los permisos que se AWS IoT puedan utilizar al aprovisionar dispositivos. Puede usar esta política administrada o crear una política personalizada que limite los permisos de la política administrada para el caso de uso.

En esta sección, se crea una función de IAM que permite AWS IoT aprovisionar recursos para los dispositivos y se crea una plantilla de aprovisionamiento de flotas que utiliza esa función de IAM.

Cómo crear una plantilla de aprovisionamiento de flotas

1. Cree una función de IAM que AWS IoT pueda asumir como aprovisionamiento de recursos en su empresa. Cuenta de AWS Haga lo siguiente:

   a. Cree un archivo que contenga el documento de política de confianza que le AWS IoT permita asumir el rol.

      Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

      ```
      nano aws-iot-trust-policy.json
      ```

      Copie el siguiente JSON en el archivo.

      JSON

      ```
      {
        "Version":"2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
      ```

```
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Cree un rol de IAM con un documento de política de confianza.

- *GreengrassFleetProvisioningRole*Sustitúyalo por el nombre del rol de IAM que se va a crear.

```
aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-policy-document file://aws-iot-trust-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": "*"
    }
  ]
}
```

c. Revise la política AWSIoTThingsde registro, que permite el acceso a todos los permisos que se AWS IoT puedan utilizar al aprovisionar dispositivos. Puede usar esta política

administrada o crear una política personalizada que defina los permisos restringidos para el caso de uso. Si decide crear una política personalizada, hágalo ahora.

d. Adjunte la política de IAM al rol de aprovisionamiento de flotas.

- Reemplace *GreengrassFleetProvisioningRole* por el nombre del rol de IAM.
- Si creó una política personalizada en el paso anterior, sustituya el ARN de la política por el ARN de la política de IAM que vaya a utilizar.

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

2. (Opcional) Cree un enlace de aprovisionamiento previo, que es una función de AWS Lambda que valida los parámetros de plantilla que los dispositivos proporcionan durante el registro. Puede usar un enlace de aprovisionamiento previo para tener más control sobre cómo y cuántos dispositivos están integrados en la Cuenta de AWS. Para obtener más información, consulte los enlaces previos al aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .
3. Cree una plantilla de aprovisionamiento de flotas. Haga lo siguiente:

a. Cree un archivo que contenga el documento de plantilla de aprovisionamiento.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano greengrass-fleet-provisioning-template.json
```

Escriba el documento de plantilla de aprovisionamiento. Puedes empezar con el siguiente ejemplo de plantilla de aprovisionamiento, que especifica la creación de AWS IoT algo con las siguientes propiedades:

- El nombre del objeto es el valor que se especifica en el parámetro de la plantilla `ThingName`.
- El objeto es una parte del grupo de objetos que se especifica en el parámetro `ThingGroupName` de plantilla. El grupo de cosas debe existir en su. Cuenta de AWS
- El certificado de la cosa lleva `GreengrassV2IoTThingPolicy` adjunto el nombre de la AWS IoT política.

Para más información, consulte Plantillas de aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

```
{
  "Parameters": {
    "ThingName": {
      "Type": "String"
    },
    "ThingGroupName": {
      "Type": "String"
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "MyThing": {
      "OverrideSettings": {
        "AttributePayload": "REPLACE",
        "ThingGroups": "REPLACE",
        "ThingTypeName": "REPLACE"
      },
      "Properties": {
        "AttributePayload": {},
        "ThingGroups": [
          {
            "Ref": "ThingGroupName"
          }
        ],
        "ThingName": {
          "Ref": "ThingName"
        }
      },
      "Type": "AWS::IoT::Thing"
    },
    "MyPolicy": {
      "Properties": {
        "PolicyName": "GreengrassV2IoTThingPolicy"
      },
      "Type": "AWS::IoT::Policy"
    },
    "MyCertificate": {
```

```
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "Active"
      },
      "Type": "AWS::IoT::Certificate"
    }
  }
}
```

> **Note**
>
> *MyThingMyPolicy*, y *MyCertificate* son nombres arbitrarios que identifican cada especificación de recurso de la plantilla de aprovisionamiento de flota. AWS IoT no utiliza estos nombres en los recursos que crea a partir de la plantilla. Puede utilizar estos nombres o sustituirlos por valores que lo ayuden a identificar cada recurso de la plantilla.

b. Cree la plantilla de aprovisionamiento de flotas a partir del documento de plantilla de aprovisionamiento.

- Sustituya *GreengrassFleetProvisioningTemplate* por el nombre de la plantilla a crear.
- Sustituya la descripción de la plantilla por una descripción de la plantilla.
- Sustituya rol del ARN de aprovisionamiento por el ARN del rol creó anteriormente.

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

> **Note**
>
> Si creó un enlace de aprovisionamiento previo, especifique el ARN de la función de Lambda del enlace de aprovisionamiento previo con el argumento `--pre-provisioning-hook`.
>
> ```
> --pre-provisioning-hook targetArn=arn:aws:lambda:us-
> west-2:123456789012:function:GreengrassPreProvisioningHook
> ```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

Creación de un certificado de notificación de aprovisionamiento y una clave privada

Los certificados de reclamación son certificados X.509 que permiten a los dispositivos registrarse como AWS IoT objetos y recuperar un certificado de dispositivo X.509 exclusivo para utilizarlo en las operaciones habituales. Tras crear un certificado de reclamación, debe adjuntar una AWS IoT política que permita a los dispositivos utilizarlo para crear certificados de dispositivos únicos y aprovisionarlos con una plantilla de aprovisionamiento de flota. Los dispositivos con el certificado de reclamación se pueden aprovisionar únicamente con la plantilla de aprovisionamiento que usted permita en la política AWS IoT .

En esta sección, usted crea el certificado de reclamación y lo configura para que los dispositivos lo usen con la plantilla de aprovisionamiento de flotas que creó en la sección anterior.

> **Important**
>
> Las claves privadas de la notificación de aprovisionamiento deben estar protegidas en todo momento, también en el dispositivo principal de Greengrass. Te recomendamos que utilices CloudWatch las estadísticas y los registros de Amazon para detectar indicios de uso indebido, como el uso no autorizado del certificado de reclamación para aprovisionar dispositivos. Si detecta un uso incorrecto, deshabilite el certificado de notificación de aprovisionamiento para que no se pueda utilizar para el aprovisionamiento de dispositivos. Para obtener más información, consulte Monitorear AWS IoT en la Guía para desarrolladores de AWS IoT Core .
>
> Para ayudarte a gestionar mejor el número de dispositivos y los dispositivos que se registran automáticamente en el tuyo Cuenta de AWS, puedes especificar un enlace previo al aprovisionamiento al crear una plantilla de aprovisionamiento de flotas. Un enlace de preaprovisionamiento es una AWS Lambda función que valida los parámetros de plantilla que proporcionan los dispositivos durante el registro. Por ejemplo, puede crear un enlace previo al aprovisionamiento que compruebe un ID de un dispositivo con una base de datos para comprobar que el dispositivo tiene permiso de aprovisionamiento. Para obtener más información, consulte los enlaces previos al aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

Cómo crear un certificado de reclamación de aprovisionamiento y una clave privada

1. Cree una carpeta en la que pueda descargar el certificado de reclamación y la clave privada.

```
mkdir claim-certs
```

2. Cree y guarde un certificado y una clave privada para usarlos en el aprovisionamiento. AWS IoT proporciona certificados de cliente firmados por la autoridad de certificación (CA) raíz de Amazon.

   Linux or Unix

   ```
   aws iot create-keys-and-certificate \
     --certificate-pem-outfile "claim-certs/claim.pem.crt" \
     --public-key-outfile "claim-certs/claim.public.pem.key" \
     --private-key-outfile "claim-certs/claim.private.pem.key" \
     --set-as-active
   ```

   Windows Command Prompt (CMD)

   ```
   aws iot create-keys-and-certificate ^
     --certificate-pem-outfile "claim-certs/claim.pem.crt" ^
     --public-key-outfile "claim-certs/claim.public.pem.key" ^
     --private-key-outfile "claim-certs/claim.private.pem.key" ^
     --set-as-active
   ```

   PowerShell

   ```
   aws iot create-keys-and-certificate `
     --certificate-pem-outfile "claim-certs/claim.pem.crt" `
     --public-key-outfile "claim-certs/claim.public.pem.key" `
     --private-key-outfile "claim-certs/claim.private.pem.key" `
     --set-as-active
   ```

   La respuesta contiene información sobre el certificado, si la solicitud se realiza correctamente. Guarde el ARN del certificado para usarlo más adelante.

3. Cree y adjunte una AWS IoT política que permita a los dispositivos usar el certificado para crear certificados de dispositivo únicos y aprovisionarlos con la plantilla de aprovisionamiento de flotas. La siguiente política permite acceder a la API MQTT de aprovisionamiento de dispositivos. Para obtener más información, consulte la API MQTT de aprovisionamiento de dispositivos en la Guía para desarrolladores de AWS IoT Core .

Haga lo siguiente:

a. Cree un archivo que contenga el documento AWS IoT de política que requieren los dispositivos principales de Greengrass.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano greengrass-provisioning-claim-iot-policy.json
   ```

   Copie el siguiente JSON en el archivo.

   - Sustituya cada instancia por la *region* instancia en la Región de AWS que configuró el aprovisionamiento de la flota.
   - Sustituya cada instancia de *account-id* por su Cuenta de AWS ID.
   - Sustituya cada instancia *GreengrassFleetProvisioningTemplate* de por el nombre de la plantilla de aprovisionamiento de flota que creó en la sección anterior.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "iot:Connect",
         "Resource": "*"
       },
       {
         "Effect": "Allow",
         "Action": [
           "iot:Publish",
           "iot:Receive"
         ],
         "Resource": [
         "arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/create/
   *",
     "arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
   templates/GreengrassFleetProvisioningTemplate/provision/*"
   ```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
      "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/
create/*",
  "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

b. Cree una AWS IoT política a partir del documento de política.

- *GreengrassProvisioningClaimPolicy*Sustitúyala por el nombre de la política que se va a crear.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

JSON

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
```

```
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
        \"arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/
create/*\",
  \"arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
        \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
certificates/create/*\",
  \"arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

4. Adjunte la AWS IoT política al certificado de notificación de aprovisionamiento.

   - *GreengrassProvisioningClaimPolicy*Sustitúyala por el nombre de la política que se va a adjuntar.
   - Sustituya el ARN de destino por el ARN del certificado de reclamación de aprovisionamiento.

   ```
   aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
   target arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Ahora tiene un certificado de notificación de aprovisionamiento y una clave privada que los dispositivos pueden usar para registrarse AWS IoT y aprovisionarse como dispositivos principales

de Greengrass. Puede incrustar el certificado de reclamación y la clave privada en los dispositivos durante la fabricación, o bien copiar el certificado y la clave en los dispositivos antes de instalar el software AWS IoT Greengrass Core. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota.

Configurar el complemento de aprovisionamiento de AWS IoT flotas

El complemento de aprovisionamiento de AWS IoT flotas proporciona los siguientes parámetros de configuración que puede personalizar al instalar el software AWS IoT Greengrass Core con el aprovisionamiento de flotas.

`rootPath`

La ruta a la carpeta que se va a utilizar como raíz del software AWS IoT Greengrass principal.

`awsRegion`

La Región de AWS que utiliza el complemento de aprovisionamiento de flotas para aprovisionar AWS recursos.

`iotDataEndpoint`

El punto final AWS IoT de datos para su. Cuenta de AWS

`iotCredentialEndpoint`

El punto final de AWS IoT credenciales para su Cuenta de AWS.

`iotRoleAlias`

El alias del AWS IoT rol que apunta a un rol de IAM de intercambio de fichas. El proveedor de AWS IoT credenciales asume esta función para permitir que el dispositivo principal de Greengrass interactúe con AWS los servicios. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

`provisioningTemplate`

La plantilla de aprovisionamiento de AWS IoT flota que se utilizará para AWS aprovisionar recursos. Esta plantilla debe especificar lo siguiente:

- Cualquier AWS IoT cosa: un recurso. Puede especificar una lista de grupos de objetos existentes para implementar componentes en cada dispositivo cuando esté en línea.
- Un recurso AWS IoT político. Este recurso puede definir al menos una de las siguientes propiedades:

- El nombre de una AWS IoT política existente. Si elige esta opción, los dispositivos principales que cree a partir de esta plantilla utilizarán la misma AWS IoT política y podrá gestionar sus permisos como una flota.
- Un documento AWS IoT de política. Si elige esta opción, cada dispositivo principal que cree a partir de esta plantilla utilizará una AWS IoT política única y podrá administrar los permisos de cada dispositivo principal individual.

- Un recurso AWS IoT de certificados. Este recurso de certificado debe usar el parámetro `AWS::IoT::Certificate::Id` para adjuntar el certificado al dispositivo principal. Para obtener más información, consulte el Just-in-time aprovisionamiento en la Guía para AWS IoT desarrolladores.

Para más información, consulte Plantillas de aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

`claimCertificatePath`

La ruta al certificado de reclamación de aprovisionamiento de la plantilla de aprovisionamiento que especifica en `provisioningTemplate`. Para obtener más información, consulta CreateProvisioningClaim en la AWS IoT Core Referencia de la API de .

`claimCertificatePrivateKeyPath`

La ruta a la clave privada del certificado de reclamación de aprovisionamiento de la plantilla de aprovisionamiento que especifica en `provisioningTemplate`. Para obtener más información, consulta CreateProvisioningClaim en la AWS IoT Core Referencia de la API de .

> **Important**
>
> Las claves privadas de la notificación de aprovisionamiento deben estar protegidas en todo momento, también en el dispositivo principal de Greengrass. Te recomendamos que utilices CloudWatch las estadísticas y los registros de Amazon para detectar indicios de uso indebido, como el uso no autorizado del certificado de reclamación para aprovisionar dispositivos. Si detecta un uso incorrecto, deshabilite el certificado de notificación de aprovisionamiento para que no se pueda utilizar para el aprovisionamiento de dispositivos. Para obtener más información, consulte Monitorear AWS IoT en la Guía para desarrolladores de AWS IoT Core .
> Para ayudarte a gestionar mejor el número de dispositivos y los dispositivos que se registran automáticamente en el tuyo Cuenta de AWS, puedes especificar un enlace previo al aprovisionamiento al crear una plantilla de aprovisionamiento de flotas. Un

enlace de preaprovisionamiento es una AWS Lambda función que valida los parámetros de plantilla que proporcionan los dispositivos durante el registro. Por ejemplo, puede crear un enlace previo al aprovisionamiento que compruebe un ID de un dispositivo con una base de datos para comprobar que el dispositivo tiene permiso de aprovisionamiento. Para obtener más información, consulte los enlaces previos al aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

`rootCaPath`

La ruta al certificado de la autoridad de certificación (CA) raíz de Amazon.

`templateParameters`

(Opcional) El mapa de parámetros que se debe proporcionar a la plantilla de aprovisionamiento de la flota. Para obtener más información, consulte la sección de parámetros de las plantillas de aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

`deviceId`

(Opcional) El identificador del dispositivo que se utilizará como ID de cliente cuando el complemento de aprovisionamiento de flotas cree una conexión MQTT a AWS IoT.

Predeterminado: un UUID con asignación al azar.

`mqttPort`

(Opcional) El puerto que se utilizará para las conexiones MQTT.

Valor predeterminado: `8883`

`proxyUrl`

(Opcional) La dirección URL del servidor proxy, en el formato `scheme://userinfo@host:port`. Para utilizar un proxy HTTPS, debe utilizar la versión 1.1.0 o posterior del complemento de aprovisionamiento de flotas.

- `scheme`: el esquema, que debe ser `http` o `https`.

> **Important**
>
> Los dispositivos principales de Greengrass deben ejecutar la versión 2.5.0 o versiones posteriores del núcleo de Greengrass para usar proxies HTTPS.

Si configura un proxy HTTPS, debe agregar el certificado de la CA del servidor proxy al certificado de la CA raíz de Amazon del dispositivo principal. Para obtener más información, consulte Permita que el dispositivo principal confíe en un proxy HTTPS.

- `userinfo`: (opcional) la información de nombre de usuario y contraseña. Si especifica esta información en `url`, el dispositivo principal de Greengrass ignora los campos `username` y `password`.
- `host`: el nombre de host o dirección IP del servidor proxy.
- `port`: (opcional) el número de puerto. Si no especifica el puerto, el dispositivo principal de Greengrass usará los siguientes valores predeterminados:
  - `http`: 80
  - `https`: 443

`proxyUserName`

(Opcional) El nombre de usuario que autentica el servidor proxy.

`proxyPassword`

(Opcional) El nombre de usuario que autentica el servidor proxy.

csrPath

(Opcional) La ruta al archivo de solicitud de firma de certificado (CSR) que se utilizará para crear el certificado del dispositivo a partir de una CSR. Para obtener más información, consulte Aprovisionamiento por reclamación en la Guía para desarrolladores de AWS IoT Core .

csrPrivateKeyRuta

(Opcional, obligatorio si se declara `csrPath`) La ruta a la clave privada utilizada para generar la CSR. La clave privada debe haberse utilizado para generar la CSR. Para obtener más información, consulte Aprovisionamiento por reclamación en la Guía para desarrolladores de AWS IoT Core .

certificatePath

(Opcional) Ruta que se utilizará para guardar el certificado del dispositivo descargado.

privateKeyPath

(Opcional) Ruta que se utilizará para guardar la clave privada del dispositivo descargado.

Registro de cambios del complemento de aprovisionamiento de flotas AWS IoT

En la tabla a continuación, se describen los cambios de cada versión del aprovisionamiento de flotas AWS IoT por complemento de reclamación (`aws.greengrass.FleetProvisioningByClaim`).

Versión	Cambios
1.2.2	Mejoras y correcciones de errores • Agregue compatibilidad con las rutas personalizadas para certificados de dispositivos principales (`certificatePath`) y claves privadas (`privateKeyPath`).
1.2.1	Mejoras y correcciones de errores • Soluciona un problema por el que el complemento de aprovisionamiento de flotas estaba fuera de línea durante el inicio del núcleo de Greengras s. El complemento de aprovisionamiento de flotas ahora reintenta indefinidamente las llamadas de conexión MQTT.
1.2.0	Mejoras y correcciones de errores • Suma compatibilidad con el aprovisionamiento de dispositivos mediante una solicitud de firma de certificado con una ruta de clave privada configurable. • Pequeñas correcciones y mejoras.
1.1.0	Mejoras y correcciones de errores • Suma compatibilidad con otros formatos de rutas de archivos al configurar el complemento en dispositivos Windows. • Suma compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red y Permita que el dispositivo principal confíe en un proxy HTTPS.
1.0.0	Versión inicial.

Instale el software AWS IoT Greengrass principal con aprovisionamiento de recursos personalizado

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass.

El instalador de software AWS IoT Greengrass Core proporciona una interfaz Java que puede implementar en un complemento personalizado que aprovisione AWS los recursos necesarios. Puede desarrollar un complemento de aprovisionamiento para usar certificados de cliente X.509 personalizados o ejecutar pasos de aprovisionamiento complejos que otros procesos de instalación no admiten. Para obtener más información, consulte Crear sus propios certificados de cliente en la Guía para desarrolladores de AWS IoT Core .

Para ejecutar un complemento de aprovisionamiento personalizado al instalar el software AWS IoT Greengrass Core, debe crear un archivo JAR que se proporciona al instalador. El instalador ejecuta el complemento y el complemento devuelve una configuración de aprovisionamiento que define los AWS recursos del dispositivo principal de Greengrass. El instalador usa esta información para configurar el software AWS IoT Greengrass principal en el dispositivo. Para obtener más información, consulte Desarrollo de complementos de aprovisionamiento personalizados.

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Requisitos previos
- Configuración del entorno del dispositivo
- Descargue el software AWS IoT Greengrass principal
- Instale el software principal AWS IoT Greengrass
- Desarrollo de complementos de aprovisionamiento personalizados

Requisitos previos

Para instalar el software AWS IoT Greengrass Core con aprovisionamiento personalizado, debe disponer de lo siguiente:

- Un archivo JAR para un complemento de aprovisionamiento personalizado que implementa la `DeviceIdentityInterface`. El complemento de aprovisionamiento personalizado debe devolver valores para cada parámetro de configuración del núcleo y del sistema. De lo contrario, debe proporcionar esos valores en el archivo de configuración durante la instalación. Para obtener más información, consulte Desarrollo de complementos de aprovisionamiento personalizados.

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

```
java -version
```

El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

   ```
   sudo visudo
   ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:

   a. Pulse la tecla Windows para abrir el menú de inicio.

   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.

   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.

   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.

   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.

f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.

3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.

4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el `wmic`comando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.

6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

   > **Note**
   >
   > En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

```
jar verified.
```

ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

```
jar is unsigned.
```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *GreengrassInstaller*Sustitúyalo por la carpeta que desee usar.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
```

```
rm greengrass-nucleus-latest.zip
```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.
> Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Instale el software principal AWS IoT Greengrass

Ejecute el instalador con argumentos que especifiquen las siguientes acciones:

- Instálelo desde un archivo de configuración parcial que especifique el uso de su complemento de aprovisionamiento personalizado para aprovisionar AWS recursos. El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica la configuración de todos los componentes de Greengrass del dispositivo. El instalador crea un archivo de configuración completo a partir del archivo de configuración parcial que usted proporciona y de los AWS recursos que crea el complemento de aprovisionamiento personalizado.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.
- Configure el software AWS IoT Greengrass principal como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

Para instalar el software AWS IoT Greengrass Core (Linux)

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *`GreengrassInstaller`*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie el siguiente contenido YAML en el archivo.

   ```
   ---
   ```

```
system:
  rootpath: "/greengrass/v2"
  # The following values are optional. Return them from the provisioning plugin or
 set them here.
  # certificateFilePath: ""
  # privateKeyPath: ""
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      # The following values are optional. Return them from the provisioning plugin
 or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

A continuación, proceda del modo siguiente:

- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- Reemplace cada instancia de */greengrass/v2* por la carpeta raíz de Greengrass.
- (Opcional) Especifique los valores de configuración del sistema y del núcleo. Debe establecer estos valores si su complemento de aprovisionamiento no los proporciona.
- (Opcional) Especifique los parámetros de configuración para proporcionarlos a su complemento de aprovisionamiento.

> **Note**
>
> En este archivo de configuración, puede personalizar otras opciones de configuración, como los puertos y el proxy de red que use, tal como se muestra en el siguiente ejemplo. Para obtener más información, consulte la Configuración del núcleo de Greengrass.
>
> ```
> ---
> system:
> ```

```
  rootpath: "/greengrass/v2"
  # The following values are optional. Return them from the provisioning
 plugin or set them here.
  # certificateFilePath: ""
  # privateKeyPath: ""
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
 plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

3. Ejecute el instalador. Especifique `--trusted-plugin` para proporcionar su complemento de aprovisionamiento personalizado y especifique `--init-config` para proporcionar el archivo de configuración.

> **Note**
>
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

- Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.
- Sustituya cada instancia de por *GreengrassInstaller* la carpeta en la que desempaquetó el instalador.
- Reemplace la ruta al archivo JAR del complemento de aprovisionamiento personalizado por la ruta al archivo JAR del complemento.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

> **Note**
>
> No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`. Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

4. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Si instaló el software AWS IoT Greengrass Core como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

Para obtener más información sobre cómo configurar y utilizar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Desarrollo de complementos de aprovisionamiento personalizados

Para desarrollar un complemento de aprovisionamiento personalizado, cree una clase Java que implemente la interfaz `com.aws.greengrass.provisioning.DeviceIdentityInterface`. Puede incluir el archivo JAR del núcleo de Greengrass en su proyecto para acceder a esta interfaz y a sus clases. Esta interfaz define un método que introduce una configuración de complemento y genera una configuración de aprovisionamiento. La configuración de aprovisionamiento define las configuraciones para el sistema y el componente núcleo de Greengrass. El instalador del software AWS IoT Greengrass principal utiliza esta configuración de aprovisionamiento para configurar el software AWS IoT Greengrass principal en un dispositivo.

Después de desarrollar un complemento de aprovisionamiento personalizado, créelo como un archivo JAR que puede proporcionar al instalador del software AWS IoT Greengrass principal para que ejecute el complemento durante la instalación. El instalador ejecuta el complemento de aprovisionamiento personalizado en la misma JVM que utiliza el instalador, por lo que puede crear un JAR que contenga únicamente el código del complemento.

> **Note**
>
> El complemento de aprovisionamiento de flota de AWS IoT implementa `DeviceIdentityInterface` para utilizar el aprovisionamiento de flota durante la instalación. El complemento de aprovisionamiento de flota es de código abierto, por lo que puede explorar su código de origen para ver un ejemplo de cómo utilizar la interfaz del complemento de aprovisionamiento. Para obtener más información, consulte el complemento de aprovisionamiento de AWS IoT flotas en. GitHub

Temas

- Requisitos
- Implemente la interfaz DeviceIdentityInterface

Requisitos

Para desarrollar un complemento de aprovisionamiento personalizado, debe crear una clase Java que cumpla con los siguientes requisitos:

- Usa el paquete `com.aws.greengrass` o un paquete dentro del paquete `com.aws.greengrass`.
- Tiene un constructor sin argumentos.
- Implementa la interfaz `DeviceIdentityInterface`. Para obtener más información, consulte Implemente la interfaz DeviceIdentityInterface.

Implemente la interfaz DeviceIdentityInterface

Para utilizar la interfaz `com.aws.greengrass.provisioning.DeviceIdentityInterface` en su complemento personalizado, agregue el núcleo de Greengrass como una dependencia a su proyecto.

Para utilizarla DeviceIdentityInterface en un proyecto de complemento de aprovisionamiento personalizado

- Puede agregar el archivo JAR del núcleo de Greengrass como biblioteca o agregar el núcleo de Greengrass como una dependencia de Maven. Realice una de las siguientes acciones:
  - Para añadir el archivo JAR del núcleo de Greengrass como biblioteca, descargue el software AWS IoT Greengrass Core, que contiene el JAR del núcleo de Greengrass. Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:
    - https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

    Puede encontrar el archivo JAR del núcleo de Greengrass (`Greengrass.jar`) en la carpeta `lib` del archivo ZIP. Agregue este archivo JAR a su proyecto.
  - Para utilizar el núcleo de Greengrass en un proyecto de Maven, agregue una dependencia al artefacto `nucleus` del grupo `com.aws.greengrass`. También debe agregar el

repositorio `greengrass-common`, ya que el núcleo de Greengrass no está disponible en el repositorio central de Maven.

```
<project ...>
    ...
    <repositories>
        <repository>
            <id>greengrass-common</id>
            <name>greengrass common</name>
            <url>https://d2jrmugq4soldf.cloudfront.net/snapshots</url>
        </repository>
    </repositories>
    ...
    <dependencies>
        <dependency>
            <groupId>com.aws.greengrass</groupId>
            <artifactId>nucleus</artifactId>
            <version>2.5.0-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

La DeviceIdentityInterface interfaz

La interfaz `com.aws.greengrass.provisioning.DeviceIdentityInterface` tiene la siguiente forma.

> **Note**
>
> También puede explorar estas clases en el paquete com.aws.greengrass.provisioning del código fuente del núcleo de Greengrass en. GitHub

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
            throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
    String name();
}
```

```
com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}

com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy;  // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

Todos los valores de configuración de `SystemConfiguration` y `NucleusConfiguration` son necesarios para instalar el software Core, pero puede devolverlos AWS IoT Greengrass . `null` Si su complemento de aprovisionamiento personalizado devuelve `null` algún valor de configuración, debe proporcionar ese valor en la configuración del sistema o del núcleo al crear el `config.yaml` archivo que se va a proporcionar al instalador del software AWS IoT Greengrass Core. Si su complemento de aprovisionamiento personalizado devuelve un valor no nulo para una opción que también ha definido en `config.yaml`, el instalador sustituirá el valor en `config.yaml` por el valor devuelto por el complemento.

Argumentos del instalador

El software AWS IoT Greengrass Core incluye un instalador que configura el software y proporciona los recursos de AWS necesarios para que se ejecute el dispositivo principal de Greengrass. El instalador incluye los siguientes argumentos que puede especificar para configurar la instalación:

`-h, --help`

(Opcional) Muestra la información de ayuda del instalador.

`--version`

(Opcional) Muestra la versión del software AWS IoT Greengrass Core.

`-Droot`

(Opcional) La ruta a la carpeta que se va a usar como la raíz del software AWS IoT Greengrass Core.

> **Note**
>
> Este argumento establece una propiedad de JVM, por lo que debe especificarla antes de `-jar` cuando ejecute el instalador. Por ejemplo, especifique `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar`.

Predeterminado:

- Linux: `~/.greengrass`
- Windows: : `%USERPROFILE%/.greengrass`

`-ar, --aws-region`

La Región de AWS que el software AWS IoT Greengrass Core usa para recuperar o crear los recursos de AWS necesarios.

`-p, --provision`

(Opcional) Puede registrar este dispositivo como un objeto AWS IoT y aprovisionar los recursos de AWS que necesite el dispositivo principal. Si especifica `true`, el software AWS IoT Greengrass Core aprovisiona un objeto AWS IoT, (opcional) un grupo de objetos AWS IoT, un rol de IAM y un alias de rol AWS IoT.

Predeterminado: `false`

`-tn, --thing-name`

(Opcional) El nombre del objeto AWS IoT que registra como este dispositivo principal. Si el objeto con ese nombre no existe en su Cuenta de AWS, será creado por el software AWS IoT Greengrass Core.

> **Note**
>
> El nombre del objeto no puede contener dos puntos (`:`).

Debe especificar `--provision true` si desea aplicar este argumento.

Valor predeterminado: `GreengrassV2IotThing_` más un UUID asignado al azar.

`-tgn`, `--thing-group-name`

(Opcional) El nombre del grupo de objetos AWS IoT al que se agrega el objeto AWS IoT de este dispositivo principal. Si una implementación se dirige a este grupo de objetos, este dispositivo principal recibe esa implementación cuando se conecta a AWS IoT Greengrass. Si el grupo de objetos con este nombre no existe en su Cuenta de AWS, el software AWS IoT Greengrass Core lo crea.

> **Note**
>
> El nombre del grupo de objetos no puede contener dos puntos (`:`).

Debe especificar `--provision true` si desea aplicar este argumento.

`-tpn`, `--thing-policy-name`

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass.

(Opcional) El nombre de la política de AWS IoT que se debe adjuntar al certificado del objeto AWS IoT de este dispositivo principal. Si la política AWS IoT con este nombre no existe en su Cuenta de AWS, el software AWS IoT Greengrass Core la crea.

El software AWS IoT Greengrass Core crea una política de AWS IoT permisiva de forma predeterminada. Puede limitar el alcance de esta política o crear una política personalizada en la que restrinja los permisos según su caso de uso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

Debe especificar `--provision true` si desea aplicar este argumento.

Valor predeterminado: `GreengrassV2IoTThingPolicy`

`-trn`, `--tes-role-name`

(Opcional) El nombre del rol de IAM que se va a usar para adquirir las credenciales de AWS que permiten al dispositivo principal interactuar con los servicios de AWS. Si el rol con este nombre no existe en su Cuenta de AWS, el software AWS IoT Greengrass Core lo crea con la política `GreengrassV2TokenExchangeRoleAccess`. Este rol no tiene acceso a los buckets de S3 donde aloja los artefactos de los componentes. Por lo tanto, debe agregar permisos a los buckets y objetos de S3 de sus artefactos al crear un componente. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Debe especificar `--provision true` si desea aplicar este argumento.

Valor predeterminado: `GreengrassV2TokenExchangeRole`

`-tra`, `--tes-role-alias-name`

(Opcional) El nombre del alias del rol AWS IoT que apunta al rol de IAM que proporciona las credenciales de AWS para este dispositivo principal. Si el alias del rol con este nombre no existe en su Cuenta de AWS, el software AWS IoT Greengrass Core lo crea y lo dirige al rol de IAM que especifique.

Debe especificar `--provision true` si desea aplicar este argumento.

Valor predeterminado: `GreengrassV2TokenExchangeRoleAlias`

`-ss`, `--setup-system-service`

(Opcional) Puede configurar el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute cuando arranque el dispositivo. El nombre del servicio del sistema es `greengrass`. Para obtener más información, consulte Configuración del núcleo de Greengrass como un servicio del sistema.

En los sistemas operativos Linux, este argumento requiere que el sistema de inicio systemd esté disponible en el dispositivo.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass Core como un servicio del sistema.

Predeterminado: `false`

`-u, --component-default-user`

El nombre o ID del usuario que el software AWS IoT Greengrass Core usa para ejecutar los componentes. Por ejemplo, puede especificar **`ggc_user`**. Este valor es obligatorio cuando se ejecuta el instalador en sistemas operativos Windows.

En los sistemas operativos Linux, también puede especificar el grupo de forma opcional. Especifique el usuario y el grupo separados por dos puntos. Por ejemplo, **`ggc_user:ggc_group`**.

Las siguientes consideraciones adicionales se aplican a los sistemas operativos Linux:

- Si se ejecuta como raíz, el usuario del componente predeterminado es el usuario definido en el archivo de configuración. Si el archivo de configuración no define un usuario, el valor predeterminado es `ggc_user:ggc_group`. Si no existen `ggc_user` o `ggc_group`, el software los crea.
- Si se ejecuta como un usuario no raíz, el software AWS IoT Greengrass Core usa ese usuario para ejecutar los componentes.
- Si no especifica un grupo, el software AWS IoT Greengrass Core usa el grupo principal del usuario del sistema.

Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

`-d, --deploy-dev-tools`

(Opcional) Puede descargar e implementar el componente de la CLI de Greengrass en este dispositivo principal. Puede usar esta herramienta para desarrollar y depurar los componentes en este dispositivo principal.

> **Important**
>
> Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Debe especificar `--provision true` si desea aplicar este argumento.

Predeterminado: `false`

`-init`, `--init-config`

(Opcional) La ruta al archivo de configuración que se usa para instalar el software AWS IoT Greengrass Core. Puede usar esta opción para configurar nuevos dispositivos principales con una configuración de núcleo específica, por ejemplo.

> **Important**
>
> El archivo de configuración que especifique se fusiona con el archivo de configuración existente en el dispositivo principal. Esto incluye los componentes y las configuraciones de los componentes del dispositivo principal. Se recomienda que el archivo de configuración solo muestre las configuraciones que está intentando cambiar.

`-tp`, `--trusted-plugin`

(Opcional) La ruta a un archivo JAR para cargarlo como un complemento de confianza. Use esta opción para brindar aprovisionamiento a los archivos JAR del complemento, por ejemplo, para instalarlo con el aprovisionamiento de flota o el aprovisionamiento personalizado, o para instalarlo con la clave privada y el certificado en un módulo de seguridad de hardware.

`-s`, `--start`

(Opcional) Puede iniciar el software AWS IoT Greengrass Core después de que se haya instalado y, de forma opcional, aprovisionar recursos.

Valor predeterminado: `true`

Ejecute el software AWS IoT Greengrass principal

Después de instalar el software AWS IoT Greengrass Core, ejecútelo para conectar el dispositivo a AWS IoT Greengrass.

Al instalar el software AWS IoT Greengrass Core, puede especificar si desea instalarlo como un servicio del sistema con systemd. Si elige esta opción, el instalador ejecutará el software automáticamente y lo configurará para que se ejecute al arrancar el dispositivo.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Temas

- Compruebe si el software AWS IoT Greengrass Core se ejecuta como un servicio del sistema
- Ejecute el software AWS IoT Greengrass Core como un servicio del sistema
- Ejecute el software AWS IoT Greengrass Core sin un servicio de sistema

Compruebe si el software AWS IoT Greengrass Core se ejecuta como un servicio del sistema

Al instalar el software AWS IoT Greengrass Core, puede especificar el `--setup-system-service true` argumento para instalar el software AWS IoT Greengrass Core como un servicio del sistema. Los dispositivos Linux requieren que el sistema systemd init configure el software AWS IoT Greengrass Core como un servicio del sistema. Si elige esta opción, el instalador ejecutará el software automáticamente y lo configurará para que se ejecute al arrancar el dispositivo. El instalador muestra el siguiente mensaje si instala correctamente el software AWS IoT Greengrass Core como un servicio del sistema.

```
Successfully set up Nucleus as a system service
```

Si ya instaló el software AWS IoT Greengrass Core y no tiene la salida del instalador, puede comprobar si el software está instalado como un servicio del sistema.

Para comprobar si el software AWS IoT Greengrass principal está instalado como un servicio del sistema

- Ejecute el siguiente comando para verificar el estado del servicio del sistema Greengrass.

  Linux or Unix (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

La respuesta es similar a la del siguiente ejemplo si el software AWS IoT Greengrass principal está instalado como un servicio del sistema y está activo.

```
# greengrass.service - Greengrass Core
   Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
 Main PID: 16107 (sh)
   CGroup: /system.slice/greengrass.service
           ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
           ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
v2/alts/current/distro/lib/Greengrass...
```

Si `systemctl greengrass.service` no lo encuentra, significa que el software AWS IoT Greengrass principal no está instalado como un servicio del sistema. Para ejecutar el software, vea Ejecute el software AWS IoT Greengrass Core sin un servicio de sistema.

Windows Command Prompt (CMD)

```
sc query greengrass
```

La respuesta es similar a la del siguiente ejemplo si el software AWS IoT Greengrass principal está instalado como un servicio de Windows y está activo.

```
SERVICE_NAME: greengrass
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

PowerShell

```
Get-Service greengrass
```

La respuesta es similar a la del siguiente ejemplo si el software AWS IoT Greengrass principal está instalado como un servicio de Windows y está activo.

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

Ejecute el software AWS IoT Greengrass Core como un servicio del sistema

Si el software AWS IoT Greengrass principal está instalado como un servicio del sistema, puede usar el administrador de servicios del sistema para iniciar, detener y administrar el software. Para obtener más información, consulte Configuración del núcleo de Greengrass como un servicio del sistema.

Para ejecutar el software AWS IoT Greengrass principal

- Ejecute el siguiente comando para iniciar el software AWS IoT Greengrass Core.

  Linux or Unix (systemd)

  ```
  sudo systemctl start greengrass.service
  ```

  Windows Command Prompt (CMD)

  ```
  sc start greengrass
  ```

  PowerShell

  ```
  Start-Service greengrass
  ```

Ejecute el software AWS IoT Greengrass Core sin un servicio de sistema

En los dispositivos Linux Core, si el software AWS IoT Greengrass Core no está instalado como un servicio del sistema, puede ejecutar el script de carga del software para ejecutarlo.

Para ejecutar el software AWS IoT Greengrass principal sin un servicio del sistema

- Ejecute el siguiente comando para iniciar el software AWS IoT Greengrass Core. Si ejecuta este comando en una terminal, debe mantener abierta la sesión de la terminal para que el software AWS IoT Greengrass principal siga funcionando.
  - Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass que utilice.

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

El software imprime el siguiente mensaje si se inicia correctamente.

```
Launched Nucleus successfully.
```

Ejecute AWS IoT Greengrass el software principal en un contenedor de Docker

AWS IoT Greengrass se puede configurar para que se ejecute en un contenedor de Docker. Docker es una plataforma que le permite crear, ejecutar, probar e implementar aplicaciones basadas en contenedores de Linux. Al ejecutar una imagen de AWS IoT Greengrass Docker, puede elegir si desea proporcionar sus AWS credenciales al contenedor de Docker y permitir que el instalador del software AWS IoT Greengrass Core aprovisione automáticamente los AWS recursos que un dispositivo principal de Greengrass necesita para funcionar. Si no desea proporcionar AWS credenciales, puede aprovisionar AWS recursos manualmente y ejecutar el software AWS IoT Greengrass Core en el contenedor de Docker.

Temas

- Plataformas compatibles y requisitos
- AWS IoT Greengrass descargas del software Docker
- Elija cómo aprovisionar los recursos AWS
- Cree la imagen del AWS IoT Greengrass contenedor a partir de un Dockerfile
- Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento automático de recursos

- Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento manual de recursos
- Solución de problemas AWS IoT Greengrass en un contenedor Docker

Plataformas compatibles y requisitos

Los ordenadores host deben cumplir los siguientes requisitos mínimos para instalar y ejecutar el software AWS IoT Greengrass principal en un contenedor de Docker:

- Sistema operativo basado en Linux con conexión a Internet.
- Docker Engine, versión 18.09 o posterior.
- (Opcional) Docker Compose versión 1.22 o posterior. Docker Compose solo es necesario si quiere usar la CLI de Docker Compose para ejecutar sus imágenes de Docker.

Para ejecutar componentes de función de Lambda dentro del contenedor de Docker, debe configurar el contenedor para cumplir con requisitos adicionales. Para obtener más información, consulte Requisitos de la función de Lambda.

Ejecución de los componentes en modo de proceso

AWS IoT Greengrass no admite la ejecución de funciones de Lambda o componentes AWS proporcionados en un entorno de ejecución aislado dentro del AWS IoT Greengrass contenedor de Docker. Debe ejecutar estos componentes en modo de proceso sin ningún tipo de aislamiento.

Al configurar un componente de la función de Lambda, defina el modo de aislamiento en Sin contenedor. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

Al implementar cualquiera de los siguientes componentes AWS proporcionados, actualice la configuración de cada componente para establecer el parámetro en. `containerMode` `NoContainer` Para obtener más información acerca de las actualizaciones de configuración, consulte Actualización de las configuraciones de los componentes.

- CloudWatch métricas
- Device Defender
- Firehose
- Adaptador de protocolo Modbus-RTU
- Amazon SNS

AWS IoT Greengrass descargas del software Docker

AWS IoT Greengrass proporciona un Dockerfile para crear una imagen de contenedor que tiene el software AWS IoT Greengrass principal y las dependencias instaladas en una imagen base de Amazon Linux 2 (x86_64). Puede modificar la imagen base del Dockerfile para que se ejecute en una arquitectura de plataforma diferente. AWS IoT Greengrass

Descargue el paquete Dockerfile desde. GitHub

El Dockerfile usa una versión anterior de Greengrass. Debe actualizar el archivo para usar la versión de Greengrass que desee. Para obtener información sobre cómo crear la imagen del AWS IoT Greengrass contenedor a partir del Dockerfile, consulte. Cree la imagen del AWS IoT Greengrass contenedor a partir de un Dockerfile

Elija cómo aprovisionar los recursos AWS

Al instalar el software AWS IoT Greengrass Core en un contenedor de Docker, puede elegir si desea aprovisionar automáticamente los AWS recursos que un dispositivo principal de Greengrass necesita para funcionar o usar los recursos que aprovisiona manualmente.

- Aprovisionamiento automático de recursos: el instalador proporciona la AWS IoT AWS IoT cosa, el grupo de cosas, la función de IAM y el alias de la AWS IoT función al ejecutar la imagen del AWS IoT Greengrass contenedor por primera vez. El instalador también puede implementar las herramientas de desarrollo locales en el dispositivo principal, de modo que usted pueda usar el dispositivo para desarrollar y probar componentes de software personalizados. Para aprovisionar estos recursos automáticamente, debe proporcionar credenciales de AWS como variables de entorno a la imagen de Docker.

  Para utilizar el aprovisionamiento automático, debe configurar la variable de entorno `PROVISION=true` de Docker y montar un archivo de credenciales para proporcionar sus credenciales de AWS al contenedor.
- Aprovisionamiento manual de recursos: si no desea proporcionar AWS credenciales al contenedor, puede aprovisionar los AWS recursos manualmente antes de ejecutar la imagen del contenedor. AWS IoT Greengrass Debe crear un archivo de configuración para proporcionar información sobre estos recursos al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker.

  Para utilizar el aprovisionamiento manual, debe configurar la variable de entorno de Docker `PROVISION=false`. La opción predeterminada es el aprovisionamiento manual.

Para obtener más información, consulte Cree la imagen del AWS IoT Greengrass contenedor a partir de un Dockerfile.

Cree la imagen del AWS IoT Greengrass contenedor a partir de un Dockerfile

AWS proporciona un Dockerfile que puede descargar y usar para ejecutar AWS IoT Greengrass el software principal en un contenedor de Docker. Los archivos Docker contienen código fuente para crear imágenes de contenedores. AWS IoT Greengrass

Antes de crear una imagen de AWS IoT Greengrass contenedor, debe configurar su Dockerfile para seleccionar la versión del software AWS IoT Greengrass principal que desea instalar. También puede configurar las variables de entorno para elegir cómo aprovisionar los recursos durante la instalación y personalizar otras opciones de instalación. En esta sección se describe cómo configurar y crear una imagen de Docker a partir de un AWS IoT Greengrass Dockerfile.

Descarga del paquete Dockerfile

Puede descargar el paquete AWS IoT Greengrass Dockerfile desde: GitHub

Repositorio Docker de AWS Greengrass

Tras descargar el paquete, extraiga el contenido a la carpeta *download-directory/*`aws-greengrass-docker-`*nucleus-version* de su computadora. El Dockerfile usa una versión anterior de Greengrass. Debe actualizar el archivo para usar la versión de Greengrass que desee.

Especifique la versión del AWS IoT Greengrass software principal

Usa el siguiente argumento de compilación en el Dockerfile para especificar la versión del software AWS IoT Greengrass principal que deseas usar en la imagen de AWS IoT Greengrass Docker. De forma predeterminada, el Dockerfile usa la última versión del software Core. AWS IoT Greengrass

`GREENGRASS_RELEASE_VERSION`

La versión del software AWS IoT Greengrass principal. De forma predeterminada, el Dockerfile descarga la última versión disponible del núcleo de Greengrass. Establezca el valor en la versión del núcleo que quiere descargar.

Configuración de las variables de entorno

Las variables de entorno le permiten personalizar la forma en que se instala el software AWS IoT Greengrass principal en el contenedor de Docker. Puede configurar las variables de entorno para la imagen de AWS IoT Greengrass Docker de varias maneras.

- Para usar las mismas variables de entorno para crear varias imágenes, configure las variables de entorno directamente en el Dockerfile.
- Si usa `docker run` para iniciar el contenedor, pase variables de entorno como argumentos en el comando o establezca las variables de entorno en un archivo de variables de entorno y, a continuación, pase el archivo como un argumento. Para obtener más información sobre cómo configurar las variables de entorno en Docker, consulte las variables de entorno en la documentación de Docker.
- Si usa `docker-compose up` para iniciar su contenedor, configure las variables de entorno en un archivo de variables de entorno y, a continuación, pase el archivo como un argumento. Para obtener más información sobre cómo configurar las variables de entorno en Compose, consulte la documentación de Docker.

Puede configurar las siguientes variables de entorno para la imagen de AWS IoT Greengrass Docker.

> **Note**
>
> No modifique la variable `TINI_KILL_PROCESS_GROUP` en el Dockerfile. Esta variable permite reenviarla `SIGTERM` a todos los miembros PIDs del grupo PID para que el software AWS IoT Greengrass principal se pueda cerrar correctamente cuando se detenga el contenedor de Docker.

`GGC_ROOT_PATH`

(Opcional) La ruta a la carpeta dentro del contenedor que se usará como raíz del software AWS IoT Greengrass Core.

Valor predeterminado: `/greengrass/v2`

`PROVISION`

(Opcional) Determina si el AWS IoT Greengrass núcleo AWS aprovisiona recursos.

- Si lo especifica`true`, el software AWS IoT Greengrass Core registra la imagen del contenedor como una AWS IoT cosa y aprovisiona los AWS recursos que requiere el dispositivo principal de Greengrass. El software AWS IoT Greengrass Core aprovisiona AWS IoT cualquier cosa, (opcional) un grupo de AWS IoT cosas, un rol de IAM y un alias de AWS IoT rol. Para obtener más información, consulte Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento automático de recursos.
- Si lo especifica`false`, debe crear un archivo de configuración para proporcionárselo al instalador AWS IoT Greengrass principal que especifique el uso de los AWS recursos y certificados que creó manualmente. Para obtener más información, consulte Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento manual de recursos.

Valor predeterminado: `false`

`AWS_REGION`

(Opcional) El Región de AWS que el software AWS IoT Greengrass principal utiliza para recuperar o crear AWS los recursos necesarios.

Predeterminado: `us-east-1`.

`THING_NAME`

(Opcional) El nombre de AWS IoT lo que se registra como este dispositivo principal. Si el elemento con este nombre no existe en el tuyo Cuenta de AWS, será creado por el software AWS IoT Greengrass Core.

Debe especificar `PROVISION=true` si desea aplicar este argumento.

Valor predeterminado: `GreengrassV2IotThing_` más un UUID asignado al azar.

`THING_GROUP_NAME`

(Opcional) El nombre del grupo de elementos al AWS IoT que se agrega este dispositivo principal. AWS IoT Si una implementación se dirige a este grupo, este y otros dispositivos principales de ese grupo reciben ese despliegue cuando se conectan AWS IoT Greengrass. Si el grupo de cosas con este nombre no existe en su empresa Cuenta de AWS, el software AWS IoT Greengrass Core lo crea.

Debe especificar `PROVISION=true` si desea aplicar este argumento.

`TES_ROLE_NAME`

(Opcional) El nombre de la función de IAM que se utilizará para adquirir AWS las credenciales que permiten al dispositivo principal de Greengrass interactuar con AWS los servicios. Si el

rol con este nombre no existe en su cuenta Cuenta de AWS, el software AWS IoT Greengrass principal lo crea con la `GreengrassV2TokenExchangeRoleAccess` política. Este rol no tiene acceso a los buckets de S3 donde aloja los artefactos de los componentes. Por lo tanto, debe agregar permisos a los buckets y objetos de S3 de sus artefactos al crear un componente. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Valor predeterminado: `GreengrassV2TokenExchangeRole`

`TES_ROLE_ALIAS_NAME`

(Opcional) El nombre del alias del AWS IoT rol que apunta al rol de IAM que proporciona AWS las credenciales para el dispositivo principal de Greengrass. Si el alias del rol con este nombre no existe en su cuenta Cuenta de AWS, el software AWS IoT Greengrass principal lo crea y lo dirige al rol de IAM que especifique.

Valor predeterminado: `GreengrassV2TokenExchangeRoleAlias`

`COMPONENT_DEFAULT_USER`

(Opcional) El nombre o ID del usuario y grupo del sistema que el software AWS IoT Greengrass principal utiliza para ejecutar los componentes. Especifique el usuario y el grupo, separados por dos puntos. El grupo es opcional. Por ejemplo, puede especificar **`ggc_user:ggc_group`** o **`ggc_user`**.

- Si se ejecuta como raíz, el usuario y el grupo que defina el archivo de configuración son de forma predeterminada. Si el archivo de configuración no define un usuario ni un grupo, el valor predeterminado es `ggc_user:ggc_group`. Si no existen `ggc_user` o `ggc_group`, el software los crea.
- Si se ejecuta como un usuario que no es root, el software AWS IoT Greengrass principal utiliza ese usuario para ejecutar los componentes.
- Si no especifica un grupo, el software AWS IoT Greengrass Core utiliza el grupo principal del usuario del sistema.

Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

`DEPLOY_DEV_TOOLS`

Define si se debe descargar e implementar el componente de la CLI de Greengrass en la imagen del contenedor. Puede usar la CLI de Greengrass para desarrollar y depurar componentes localmente.

> **Important**
>
> Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Valor predeterminado: `false`

`INIT_CONFIG`

(Opcional) La ruta al archivo de configuración que se utilizará para instalar el software AWS IoT Greengrass principal. Puede usar esta opción para configurar nuevos dispositivos principales de Greengrass con una configuración de núcleo específica o para especificar recursos aprovisionados manualmente, por ejemplo. Debe montar el archivo de configuración en la ruta que especifique en este argumento.

`TRUSTED_PLUGIN`

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass.

(Opcional) La ruta a un archivo JAR para cargarlo como un complemento de confianza. Use esta opción para aprovisionar archivos JAR, como instalar con el aprovisionamiento de flota o el aprovisionamiento personalizado.

`THING_POLICY_NAME`

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass.

(Opcional) El nombre de la AWS IoT política que se va a adjuntar al certificado AWS IoT Thing de este dispositivo principal. Si la AWS IoT política con este nombre no existe en su software AWS IoT Greengrass Core Cuenta de AWS la crea.

Debe especificar `PROVISION=true` si desea aplicar este argumento.

> **Note**
>
> El software AWS IoT Greengrass Core crea una AWS IoT política permisiva de forma predeterminada. Puede limitar el alcance de esta política o crear una política

personalizada en la que restrinja los permisos según su caso de uso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

Especificación de las dependencias que desea instalar

La instrucción RUN del AWS IoT Greengrass Dockerfile prepara el entorno del contenedor para ejecutar el instalador del AWS IoT Greengrass software Core. Puede personalizar las dependencias que se instalan antes de que el instalador del software AWS IoT Greengrass Core se ejecute en el contenedor de Docker.

Cree la imagen AWS IoT Greengrass

Usa el AWS IoT Greengrass Dockerfile para crear una imagen de AWS IoT Greengrass contenedor. Puede usar la CLI de Docker o la CLI de Docker Compose para crear la imagen e iniciar el contenedor. También puede usar la CLI de Docker para crear la imagen y, a continuación, usar Docker Compose para iniciar el contenedor a partir de esa imagen.

Docker

1. En la máquina host, ejecute el siguiente comando para cambiar al directorio que contiene el Dockerfile configurado.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. Ejecute el siguiente comando para crear la imagen del AWS IoT Greengrass contenedor a partir del Dockerfile.

   ```
   sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
   ```

Docker Compose

1. En la máquina host, ejecute el siguiente comando para cambiar al directorio que contiene el Dockerfile y el archivo Compose.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. Ejecuta el siguiente comando para usar el archivo Compose para crear la imagen del AWS IoT Greengrass contenedor.

```
docker-compose -f docker-compose.yml build
```

Ha creado correctamente la imagen del AWS IoT Greengrass contenedor. La imagen de Docker tiene instalado el software AWS IoT Greengrass principal. Ahora puede ejecutar el software AWS IoT Greengrass principal en un contenedor de Docker.

Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento automático de recursos

En este tutorial, se muestra cómo instalar y ejecutar el software AWS IoT Greengrass principal en un contenedor Docker con AWS recursos aprovisionados automáticamente y herramientas de desarrollo local. Puedes usar este entorno de desarrollo para explorar las AWS IoT Greengrass funciones de un contenedor Docker. El software requiere las credenciales de AWS para aprovisionar estos recursos e implementar las herramientas de desarrollo local.

Si no puede proporcionar AWS credenciales al contenedor, puede aprovisionar los AWS recursos que el dispositivo principal necesita para funcionar. También puede implementar las herramientas de desarrollo en un dispositivo principal para usarlas como dispositivo de desarrollo. Esto le permite conceder menos permisos al dispositivo al ejecutar el contenedor. Para obtener más información, consulte Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento manual de recursos.

Requisitos previos

Necesitará lo siguiente para completar este tutorial.

- Un Cuenta de AWS. Si no dispone de una, consulte Configura un Cuenta de AWS.
- Un usuario de AWS IAM con permisos para aprovisionar los recursos AWS IoT de IAM para un dispositivo principal de Greengrass. El instalador del software AWS IoT Greengrass principal utiliza sus AWS credenciales para aprovisionar automáticamente estos recursos. Para obtener información sobre la política de IAM mínima para aprovisionar recursos automáticamente, consulte Política de IAM mínima para que el instalador aprovisione recursos.
- Una imagen de AWS IoT Greengrass Docker. Puede crear una imagen a partir del AWS IoT Greengrass Dockerfile.

- El equipo host en el que ejecute el contenedor de Docker debe cumplir los siguientes requisitos:
  - Sistema operativo basado en Linux con conexión a Internet.
  - Docker Engine, versión 18.09 o posterior.
  - (Opcional) Docker Compose versión 1.22 o posterior. Docker Compose solo es necesario si quiere usar la CLI de Docker Compose para ejecutar sus imágenes de Docker.

Configura tus credenciales AWS

En este paso, crea un archivo de credenciales en la computadora host que contiene sus credenciales de seguridad de AWS . Al ejecutar la imagen de AWS IoT Greengrass Docker, debe montar la carpeta que contiene este archivo de credenciales `/root/.aws/` en el contenedor de Docker. El AWS IoT Greengrass instalador usa estas credenciales para aprovisionar recursos en su. Cuenta de AWS Para obtener información sobre la política de IAM mínima que requiere el instalador para aprovisionar recursos automáticamente, consulte Política de IAM mínima para que el instalador aprovisione recursos.

1. Recuperación de uno de los siguientes elementos.
   - Credenciales a largo plazo para un usuario de IAM. Para obtener información sobre cómo recuperar credenciales a largo plazo, consulte Administrar claves de acceso para usuarios de IAM en la Guía del usuario de IAM.
   - (Recomendado) Credenciales temporales para un rol de IAM. Para obtener información sobre cómo obtener credenciales temporales, consulte Uso de credenciales de seguridad temporales con la AWS CLI en la Guía del usuario de IAM.
2. Cree una carpeta en la que coloque el archivo de credenciales.

   ```
   mkdir ./greengrass-v2-credentials
   ```
3. Utilice un editor de texto para crear un archivo de configuración denominado `credentials` en la carpeta `./greengrass-v2-credentials`.

   Por ejemplo, puede ejecutar el comando siguiente para usar GNU nano para crear el archivo `credentials`.

   ```
   nano ./greengrass-v2-credentials/credentials
   ```
4. Añada sus AWS credenciales al `credentials` archivo en el siguiente formato.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
 = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

Incluya `aws_session_token` solo para credenciales temporales.

> **Important**
>
> Elimine el archivo de credenciales de la computadora host después de iniciar el AWS IoT Greengrass contenedor. Si no elimina el archivo de credenciales, sus AWS credenciales permanecerán montadas dentro del contenedor. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal en un contenedor.

Creación de un archivo de entorno

En este tutorial, se utiliza un archivo de entorno para configurar las variables de entorno que se transferirán al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker. También puede utilizar el argumento `-e` o `--env` en su comando `docker run` para establecer variables de entorno en el contenedor de Docker o puede establecer las variables en un bloque `environment` en el archivo `docker-compose.yml`.

1. Utilice un editor de texto para crear un archivo de política llamado `.env`.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para crearlo en el `.env` en el directorio actual.

   ```
   nano .env
   ```

2. Copie el contenido siguiente en el archivo.

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=true
   THING_NAME=MyGreengrassCore
   THING_GROUP_NAME=MyGreengrassCoreGroup
   ```

```
TES_ROLE_NAME=GreengrassV2TokenExchangeRole
TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
```

Reemplace los siguientes valores.

- */greengrass/v2*. La carpeta raíz de Greengrass que quiera usar para la instalación. Puede establecer este valor mediante la variable de entorno `GGC_ROOT`.
- *region*. El Región de AWS lugar donde creaste los recursos.
- *MyGreengrassCore*. El nombre de la AWS IoT cosa. Si el objeto no existe, el instalador la crea. El instalador descarga los certificados para autenticarse como tal AWS IoT .
- *MyGreengrassCoreGroup*. El nombre del grupo de AWS IoT cosas. Si el grupo de objetos no existe, el instalador lo crea y le agrega un objeto. Si el grupo de objetos existe y tiene una implementación activa, el dispositivo principal descarga y ejecuta el software que especifique la implementación.
- *GreengrassV2TokenExchangeRole*. Sustitúyalo por el nombre de la función de intercambio de fichas de IAM que permite al dispositivo principal de Greengrass obtener AWS credenciales temporales. Si el rol no existe, el instalador lo crea y crea y adjunta una política denominada Access. *GreengrassV2TokenExchangeRole* Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.
- *GreengrassCoreTokenExchangeRoleAlias*. El alias del rol de intercambio de fichas. Si el alias del rol no existe, el instalador lo crea y lo dirige al rol de intercambio de token de IAM que especifique. Para obtener más información, consulte

> **Note**
>
> Puede configurar la variable de entorno `DEPLOY_DEV_TOOLS` a `true` para implementar el componente CLI de Greengrass, que le permite desarrollar componentes personalizados dentro del contenedor de Docker. Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Ejecute el software AWS IoT Greengrass principal en un contenedor

Este tutorial le muestra cómo iniciar la imagen de Docker que creó en un contenedor de Docker. Puede usar la CLI de Docker o la CLI de Docker Compose para ejecutar la imagen del software AWS IoT Greengrass principal en un contenedor de Docker.

Docker

1. Ejecute el siguiente comando para iniciar el contenedor de Docker.

   ```
   docker run --rm --init -it --name docker-image \
    -v path/to/greengrass-v2-credentials:/root/.aws/:ro \
    --env-file .env \
    -p 8883 \
    your-container-image:version
   ```

   Este comando de ejemplo usa los siguientes argumentos para ejecutar docker:

   - `--rm`. Limpia el contenedor al salir.
   - `--init`. Utiliza un proceso de inicio en el contenedor.

     > **Note**
     >
     > El `--init` argumento es necesario para cerrar el software AWS IoT Greengrass Core al detener el contenedor de Docker.

   - `-it`. (Opcional) Ejecuta el contenedor de Docker en primer plano como un proceso interactivo. En su lugar, puede sustituirlo por el argumento `-d` para ejecutar el contenedor de Docker en modo separado. Para obtener más información, consulte Separado frente a Primer plano en la documentación de Docker.
   - `--name`. Ejecuta un contenedor llamado `aws-iot-greengrass`
   - `-v`. Monta un volumen en el contenedor de Docker para que el archivo de configuración y los archivos de certificado estén disponibles para su AWS IoT Greengrass ejecución dentro del contenedor.
   - `--env-file`. (Opcional) Especifica el archivo de entorno para establecer las variables de entorno que se pasarán al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker. Este argumento solo es necesario si ha creado un archivo de entorno para establecer variables de entorno. Si no creó un archivo de entorno, puede usar

argumentos `--env` para establecer las variables de entorno directamente en el comando run de Docker.

- `-p`. (Opcional) Publica el puerto contenedor 8883 en la máquina host. Este argumento es obligatorio si desea conectarse y comunicarse a través de MQTT, ya que AWS IoT Greengrass utiliza el puerto 8883 para el tráfico de MQTT. Para abrir otros puertos, utilice argumentos adicionales `-p`.

> **Note**
>
> Para ejecutar su contenedor de Docker con mayor seguridad, puede usar los argumentos `--cap-drop` y `--cap-add` para habilitar de forma selectiva las capacidades de Linux para su contenedor. Para obtener más información, consulte Privilegios en tiempo de ejecución y capacidades de Linux en el sitio web de Docker.

2. Elimine las credenciales del `./greengrass-v2-credentials` en el dispositivo host.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Va a eliminar estas credenciales porque proporcionan amplios permisos que el dispositivo principal solo necesita durante la configuración. Si no elimina estas credenciales, los componentes de Greengrass y otros procesos que se ejecutan en el contenedor podrán acceder a ellas. Si necesita proporcionar AWS credenciales a un componente de Greengrass, utilice el servicio de intercambio de fichas. Para obtener más información, consulte Interacción con servicios de AWS.

Docker Compose

1. Utilice un editor de texto para crear un archivo Docker Compose llamado `docker-compose.yml`.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para crearlo en el `docker-compose.yml` en el directorio actual.

```
nano docker-compose.yml
```

> **Note**
>
> También puede descargar y usar la última versión del archivo Compose AWS proporcionado desde. GitHub

2. Agregue el siguiente contenido al archivo Compose. El archivo debe ser similar al siguiente ejemplo. *docker-image*Sustitúyelo por el nombre de la imagen de Docker.

```
version: '3.7'

services:
  greengrass:
    init: true
    container_name: aws-iot-greengrass
    image: docker-image
    volumes:
      - ./greengrass-v2-credentials:/root/.aws/:ro
    env_file: .env
    ports:
      - "8883:8883"
```

Los siguientes parámetros del archivo Compose de ejemplo son opcionales:

- `ports`: (Opcional) Publica el puerto contenedor 8883 en la máquina host. Este parámetro es obligatorio si desea conectarse y comunicarse a través de MQTT, ya que AWS IoT Greengrass utiliza el puerto 8883 para el tráfico de MQTT.
- `env_file`—Especifica el archivo de entorno para establecer las variables de entorno que se pasarán al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker. Este parámetro solo es necesario si ha creado un archivo de entorno para establecer variables de entorno. Si no creó un archivo de entorno, puede usar el parámetro de entorno para configurar las variables directamente en su archivo Compose.

> **Note**
>
> Para ejecutar su contenedor de Docker con mayor seguridad, puede usar `cap_drop` y `cap_add` en su archivo Compose para habilitar de forma selectiva las capacidades de Linux para su contenedor. Para obtener más información, consulte Privilegios en tiempo de ejecución y capacidades de Linux en el sitio web de Docker.

3. Ejecute el siguiente comando para iniciar el contenedor de Docker.

```
docker-compose -f docker-compose.yml up
```

4. Elimine las credenciales del `./greengrass-v2-credentials` en el dispositivo host.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Va a eliminar estas credenciales porque proporcionan amplios permisos que el dispositivo principal solo necesita durante la configuración. Si no elimina estas credenciales, los componentes de Greengrass y otros procesos que se ejecutan en el contenedor podrán acceder a ellas. Si necesita proporcionar AWS credenciales a un componente de Greengrass, utilice el servicio de intercambio de fichas. Para obtener más información, consulte Interacción con servicios de AWS.

Siguientes pasos

AWS IoT Greengrass El software principal se ejecuta ahora en un contenedor de Docker. Ejecute el siguiente comando para recuperar el ID del contenedor en el contenedor de ejecución actual.

```
docker ps
```

A continuación, puede ejecutar el siguiente comando para acceder al contenedor y explorar el software AWS IoT Greengrass principal que se ejecuta dentro del contenedor.

```
docker exec -it container-id /bin/bash
```

Para obtener más información sobre cómo crear un componente simple, consulte Paso 4: Desarrollo y prueba de un componente en su dispositivo en Tutorial: Introducción a AWS IoT Greengrass V2

> **Note**
>
> Cuando utiliza `docker exec` para ejecutar comandos dentro del contenedor de Docker, esos comandos no se capturan en los registros de Docker. Para registrar los comandos en los registros de Docker, asocie un intérprete de comandos interactivo al contenedor de Docker. Para obtener más información, consulte Asociación de un intérprete de comandos interactivo a un contenedor de Docker.

El archivo de registro AWS IoT Greengrass Core se llama `greengrass.log` y se encuentra en*/greengrass/v2*`/logs`. Los archivos de registro de componentes también se encuentran en el mismo directorio. Para copiar los registros de Greengrass en un directorio temporal del host, ejecute el siguiente comando:

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Si desea conservar los registros después de que un contenedor se cierre o se haya eliminado, le recomendamos que monte únicamente el directorio */greengrass/v2*`/logs` en el directorio de registros temporales del host, en lugar de montar todo el directorio de Greengrass. Para obtener más información, consulte Conservación de los registros de Greengrass fuera del contenedor de Docker.

Para detener un contenedor AWS IoT Greengrass Docker en ejecución, ejecute `docker stop` o`docker-compose -f docker-compose.yml stop`. Esta acción envía `SIGTERM` al proceso de Greengrass y cierra todos los procesos asociados que se iniciaron en el contenedor. El contenedor de Docker se inicializa con el ejecutable `docker-init` como PID 1, lo que ayuda a eliminar los procesos zombis restantes. Para obtener más información, consulte la sección Especificar un proceso de instalación de la documentación de Docker.

Para obtener más información acerca de la solución de problemas relacionados con la ejecución de AWS IoT Greengrass en un contenedor de Docker, consulte Solución de problemas AWS IoT Greengrass en un contenedor Docker.

Se ejecuta AWS IoT Greengrass en un contenedor Docker con aprovisionamiento manual de recursos

En este tutorial, se muestra cómo instalar y ejecutar el software AWS IoT Greengrass principal en un contenedor Docker con recursos aprovisionados manualmente. AWS

Temas

- Requisitos previos
- Recupere los puntos finales AWS IoT
- Crea cualquier AWS IoT cosa
- Creación del certificado del objeto
- Creación de un rol de intercambio de token
- Descarga de certificados al dispositivo
- Creación de un archivo de configuración
- Creación de un archivo de entorno
- Ejecute el software AWS IoT Greengrass principal en un contenedor
- Siguientes pasos

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Un. Cuenta de AWS Si no dispone de una, consulte Configura un Cuenta de AWS.
- Una imagen de AWS IoT Greengrass Docker. Puede crear una imagen a partir del AWS IoT Greengrass Dockerfile.
- El equipo host en el que ejecute el contenedor de Docker debe cumplir los siguientes requisitos:
  - Sistema operativo basado en Linux con conexión a Internet.
  - Docker Engine, versión 18.09 o posterior.
  - (Opcional) Docker Compose versión 1.22 o posterior. Docker Compose solo es necesario si quiere usar la CLI de Docker Compose para ejecutar sus imágenes de Docker.

Recupere los puntos finales AWS IoT

Obtenga los AWS IoT puntos finales que desee y guárdelos para usarlos más adelante. Cuenta de AWS El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Haga lo siguiente:

1. Obtenga el punto final AWS IoT de datos para su. Cuenta de AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenga el punto final de AWS IoT credenciales para su Cuenta de AWS.

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

Crea cualquier AWS IoT cosa

AWS IoT las cosas representan dispositivos y entidades lógicas a las que se conectan AWS IoT. Los dispositivos principales de Greengrass son AWS IoT cosas. Cuando registras un dispositivo como una AWS IoT cosa, ese dispositivo puede usar un certificado digital para autenticarse. AWS

En esta sección, crearás AWS IoT algo que represente tu dispositivo.

Para crear cualquier AWS IoT cosa

1. Crea cualquier AWS IoT cosa para tu dispositivo. En su equipo de desarrollo, ejecute el siguiente comando.

- *MyGreengrassCore*Sustitúyalo por el nombre de la cosa que se va a utilizar. Este nombre también es el nombre de su dispositivo principal de Greengrass.

> **Note**
>
> El nombre del objeto no puede contener dos puntos (`:`).

```
aws iot create-thing --thing-name MyGreengrassCore
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Opcional) Añada la AWS IoT cosa a un grupo de cosas nuevo o existente. Los grupos de objetos se usan para administrar las flotas de dispositivos principales de Greengrass. Al implementar componentes de software en sus dispositivos, puede dirigirlos a dispositivos individuales o a grupos de dispositivos. Puede agregar un dispositivo a un grupo de objetos con una implementación activa de Greengrass para implementar los componentes de software de ese grupo de objetos en el dispositivo. Haga lo siguiente:

   a. (Opcional) Cree un grupo de AWS IoT cosas.

      - *MyGreengrassCoreGroup*Sustitúyalo por el nombre del grupo de cosas que desee crear.

> **Note**
>
> El nombre del grupo de objetos no puede contener dos puntos (`:`).

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. Añada la AWS IoT cosa a un grupo de cosas.

- *MyGreengrassCore*Sustitúyala por el nombre de la AWS IoT cosa.
- *MyGreengrassCoreGroup*Sustitúyalo por el nombre del grupo de cosas.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Creación del certificado del objeto

Al registrar un dispositivo como una AWS IoT cosa, ese dispositivo puede utilizar un certificado digital para autenticarse AWS. Este certificado permite que el dispositivo se comunique con AWS IoT y AWS IoT Greengrass.

En esta sección, puede crear y descargar certificados que el dispositivo puede usar para conectarse a AWS.

Creación del certificado del objeto

1. Crea una carpeta donde descargues los certificados de la AWS IoT cosa.

```
mkdir greengrass-v2-certs
```

2. Crea y descarga los certificados de la AWS IoT cosa.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
```

```
"
  }
}
```

Guarde el nombre de recurso de Amazon (ARN) del certificado para usarlo para configurar el certificado más adelante.

A continuación, configure el certificado de la cosa. Para obtener más información, consulte Configuración del certificado del objeto.

Creación de un rol de intercambio de token

Los dispositivos principales de Greengrass utilizan una función de servicio de IAM, denominada función de intercambio de fichas, para autorizar las llamadas a los servicios. AWS El dispositivo utiliza el proveedor de AWS IoT credenciales para obtener AWS credenciales temporales para esta función, lo que permite al dispositivo interactuar con Amazon Logs AWS IoT, enviar registros a Amazon CloudWatch Logs y descargar artefactos de componentes personalizados de Amazon S3. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Se utiliza un alias de AWS IoT rol para configurar el rol de intercambio de fichas para los dispositivos principales de Greengrass. Los alias de rol le permiten cambiar el rol de intercambio de token de un dispositivo, pero mantener la configuración del dispositivo igual. Para obtener más información, consulte Autorización de llamadas a los servicios de AWS en la Guía para desarrolladores de AWS IoT Core .

En esta sección, creará un rol de IAM de intercambio de tokens y un alias de AWS IoT rol que apunte al rol. Si ya ha configurado un dispositivo principal de Greengrass, puede usar su rol de intercambio de token y su alias de rol en lugar de crear otros nuevos. A continuación, configure el objeto AWS IoT del dispositivo para que use ese rol y ese alias.

Creación de un rol de IAM de intercambio de token

1. Creación de un rol de IAM que su dispositivo puede usar como rol de intercambio de token. Haga lo siguiente:

   a. Creación de un archivo que contenga el documento de política de confianza que requiere el rol de intercambio de token.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano device-role-trust-policy.json
```

Copie el siguiente JSON en el archivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Creación del rol de intercambio de token con el documento de política de confianza.

- *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre del rol de IAM que se va a crear.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
```

```
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Creación de un archivo que contenga el documento de política de acceso que requiere el rol de intercambio de token.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano device-role-access-policy.json
```

   Copie el siguiente JSON en el archivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Esta política de acceso no permite el acceso a los artefactos de componentes en los buckets de S3. Para implementar componentes personalizados que definan artefactos en Amazon S3, debe agregar permisos al rol para permitir que su dispositivo principal recupere artefactos de componentes. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.
> Si aún no tiene un bucket de S3 para los artefactos de los componentes, puede agregar estos permisos más adelante, después de crear un bucket.

d. Creación de la política de IAM a partir del documento de política.

- *GreengrassV2TokenExchangeRoleAccess*Sustitúyalo por el nombre de la política de IAM que se va a crear.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Adjunte la política de IAM al rol de intercambio de token.

- Reemplace *GreengrassV2TokenExchangeRole* por el nombre del rol de IAM.
- Reemplace el ARN de la política por el ARN de la política de IAM que creó en el paso anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

2. Cree un alias de AWS IoT rol que apunte al rol de intercambio de fichas.

- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias del rol que se va a crear.
- Reemplace el ARN del rol por el ARN del rol de IAM que creó en el paso anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para crear un alias de rol, debe tener el permiso para transferir el rol de IAM de intercambio de token a AWS IoT. Si recibe un mensaje de error al intentar crear un alias de rol, compruebe que el AWS usuario tiene este permiso. Para obtener más información, consulte Conceder permisos a un usuario para transferir un rol a un AWS servicio en la Guía del AWS Identity and Access Management usuario.

3. Cree y adjunte una AWS IoT política que permita a su dispositivo principal de Greengrass utilizar el alias del rol para asumir el rol de intercambio de fichas. Si ya ha configurado un dispositivo

principal de Greengrass, puede adjuntar su AWS IoT política de alias de rol en lugar de crear una nueva. Haga lo siguiente:

a. (Opcional) Cree un archivo que contenga el documento AWS IoT de política que requiere el alias del rol.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano greengrass-v2-iot-role-alias-policy.json
   ```

   Copie el siguiente JSON en el archivo.

   - Reemplace el ARN del recurso por el ARN del alias de rol.

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "iot:AssumeRoleWithCertificate",
         "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
   GreengrassCoreTokenExchangeRoleAlias"
       }
     ]
   }
   ```

b. Cree una AWS IoT política a partir del documento de política.

   - *GreengrassCoreTokenExchangeRoleAliasPolicy*Sustitúyala por el nombre de la AWS IoT política que se va a crear.

   ```
   aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
    --policy-document file://greengrass-v2-iot-role-alias-policy.json
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
   ```

```
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17       \\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Adjunta la AWS IoT política al certificado de la AWS IoT cosa.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Sustitúyala por el nombre de la AWS IoT política de alias del rol.
- Reemplace el ARN de destino por el ARN del certificado de su objeto AWS IoT .

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Descarga de certificados al dispositivo

Anteriormente, descargó el certificado de su dispositivo en su computadora de desarrollo. En esta sección, puede descargar el certificado de autoridad de certificación (CA) raíz de Amazon. A continuación, si planea ejecutar el software AWS IoT Greengrass principal de Docker en un equipo diferente al de desarrollo, debe copiar los certificados en ese equipo host. El software AWS IoT Greengrass Core usa estos certificados para conectarse al servicio AWS IoT en la nube.

Descarga de certificados al dispositivo

1. En tu ordenador de desarrollo, descarga el certificado de la autoridad de certificación raíz (CA) de Amazon. AWS IoT los certificados están asociados al certificado de CA raíz de Amazon de forma predeterminada.

   Linux or Unix

   ```
   sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://
   www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .
   \greengrass-v2-certs\AmazonRootCA1.pem
   ```

2. Si planea ejecutar el software AWS IoT Greengrass principal de Docker en un dispositivo diferente al de su ordenador de desarrollo, copie los certificados en el ordenador host. Si SSH y SCP están habilitados en la computadora de desarrollo y en la computadora host, puede utilizar el comando `scp` de la computadora de desarrollo para transferir el certificado. *`device-ip-address`*Sustitúyalos por la dirección IP de la computadora host.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

Creación de un archivo de configuración

1. En la computadora host, cree una carpeta en la que coloque el archivo de configuración.

   ```
   mkdir ./greengrass-v2-config
   ```

2. Utilice un editor de texto para crear un archivo de configuración denominado `config.yaml` en la carpeta `./greengrass-v2-config`.

Por ejemplo, puede ejecutar el comando siguiente para usar GNU nano para crear el `config.yaml`.

```
nano ./greengrass-v2-config/config.yaml
```

3. Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema y los parámetros del núcleo de Greengrass.

```
---
system:
  certificateFilePath: "/tmp/certs/device.pem.crt"
  privateKeyPath: "/tmp/certs/private.pem.key"
  rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "nucleus-version"
    configuration:
      awsRegion: "region"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
```

Reemplace los siguientes valores:

- */tmp/certs*. El directorio del contenedor de Docker en el que se montan los certificados descargados al iniciar el contenedor.
- */greengrass/v2*. La carpeta raíz de Greengrass que quiera usar para la instalación. Puede establecer este valor mediante la variable de entorno `GGC_ROOT`.
- *MyGreengrassCore*. El nombre de la AWS IoT cosa.
- *nucleus-version*. La versión del software AWS IoT Greengrass principal que se va a instalar. Este valor debe coincidir con la versión de la imagen de Docker o del Dockerfile que ha descargado. Si ha descargado la imagen de Docker de Greengrass con la etiqueta `latest`, use **`docker inspect`** ***`image-id`*** para ver la versión de la imagen.
- *region*. El Región de AWS lugar donde creaste tus AWS IoT recursos. También debe especificar el mismo valor para la variable de entorno `AWS_REGION` en el archivo de entorno.

- *GreengrassCoreTokenExchangeRoleAlias*. El alias del rol de intercambio de fichas.
- *device-data-prefix*. El prefijo del punto final AWS IoT de datos.
- *device-credentials-prefix*. El prefijo del punto final de tus AWS IoT credenciales.

Creación de un archivo de entorno

En este tutorial, se utiliza un archivo de entorno para configurar las variables de entorno que se transferirán al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker. También puede utilizar el argumento `-e` o `--env` en su comando `docker run` para establecer variables de entorno en el contenedor de Docker o puede establecer las variables en un bloque `environment` en el archivo `docker-compose.yml`.

1. Utilice un editor de texto para crear un archivo de política llamado `.env`.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para crearlo en el `.env` en el directorio actual.

   ```
   nano .env
   ```

2. Copie el contenido siguiente en el archivo.

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=false
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   INIT_CONFIG=/tmp/config/config.yaml
   ```

   Reemplace los siguientes valores.

   - */greengrass/v2*. La ruta a la carpeta raíz que se va a utilizar para instalar el software AWS IoT Greengrass principal.
   - *region*. El Región de AWS lugar donde creaste tus AWS IoT recursos. Debe especificar el mismo valor para el parámetro `awsRegion` de configuración en el archivo de configuración.
   - */tmp/config/*. La carpeta en la que se monta el archivo de configuración al iniciar el contenedor de Docker.

> **Note**
>
> Puede configurar la variable de entorno `DEPLOY_DEV_TOOLS` a `true` para implementar el componente CLI de Greengrass, que le permite desarrollar componentes personalizados dentro del contenedor de Docker. Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Ejecute el software AWS IoT Greengrass principal en un contenedor

Este tutorial le muestra cómo iniciar la imagen de Docker que creó en un contenedor de Docker. Puede usar la CLI de Docker o la CLI de Docker Compose para ejecutar la imagen del software AWS IoT Greengrass principal en un contenedor de Docker.

Docker

- Este tutorial le muestra cómo iniciar la imagen de Docker que creó en un contenedor de Docker.

  ```
  docker run --rm --init -it --name docker-image \
   -v path/to/greengrass-v2-config:/tmp/config/:ro \
   -v path/to/greengrass-v2-certs:/tmp/certs:ro \
   --env-file .env \
   -p 8883 \
   your-container-image:version
  ```

  Este comando de ejemplo usa los siguientes argumentos para ejecutar docker:

  - `--rm`. Limpia el contenedor al salir.
  - `--init`. Utiliza un proceso de inicio en el contenedor.

> **Note**
>
> El `--init` argumento es necesario para cerrar el software AWS IoT Greengrass Core al detener el contenedor de Docker.

- `-it`. (Opcional) Ejecuta el contenedor de Docker en primer plano como un proceso interactivo. En su lugar, puede sustituirlo por el argumento `-d` para ejecutar el contenedor de Docker en modo separado. Para obtener más información, consulte Separado frente a Primer plano en la documentación de Docker.
- `--name`. Ejecuta un contenedor llamado `aws-iot-greengrass`
- `-v`. Monta un volumen en el contenedor de Docker para que el archivo de configuración y los archivos de certificado estén disponibles para su AWS IoT Greengrass ejecución dentro del contenedor.
- `--env-file`. (Opcional) Especifica el archivo de entorno para establecer las variables de entorno que se pasarán al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker. Este argumento solo es necesario si ha creado un archivo de entorno para establecer variables de entorno. Si no creó un archivo de entorno, puede usar argumentos `--env` para establecer las variables de entorno directamente en el comando run de Docker.
- `-p`. (Opcional) Publica el puerto contenedor 8883 en la máquina host. Este argumento es obligatorio si desea conectarse y comunicarse a través de MQTT, ya que AWS IoT Greengrass utiliza el puerto 8883 para el tráfico de MQTT. Para abrir otros puertos, utilice argumentos adicionales `-p`.

> **Note**
>
> Para ejecutar su contenedor de Docker con mayor seguridad, puede usar los argumentos `--cap-drop` y `--cap-add` para habilitar de forma selectiva las capacidades de Linux para su contenedor. Para obtener más información, consulte Privilegios en tiempo de ejecución y capacidades de Linux en el sitio web de Docker.

Docker Compose

1. Utilice un editor de texto para crear un archivo Docker Compose llamado `docker-compose.yml`.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para crearlo en el `docker-compose.yml` en el directorio actual.

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > También puedes descargar y usar la última versión del archivo Compose AWS proporcionado desde. GitHub

2. Agregue el siguiente contenido al archivo Compose. El archivo debe ser similar al siguiente ejemplo. *`your-container-name:version`*Sustitúyelo por el nombre de la imagen de Docker.

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       build:
         context: .
       container_name: aws-iot-greengrass
       image: your-container-name:version
       volumes:
         - /path/to/greengrass-v2-config:/tmp/config/:ro
         - /path/to/greengrass-v2-certs:/tmp/certs:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

   Los siguientes parámetros del archivo Compose de ejemplo son opcionales:

   - `ports`: (Opcional) Publica el puerto contenedor 8883 en la máquina host. Este parámetro es obligatorio si desea conectarse y comunicarse a través de MQTT, ya que AWS IoT Greengrass utiliza el puerto 8883 para el tráfico de MQTT.

- `env_file`—Especifica el archivo de entorno para establecer las variables de entorno que se pasarán al instalador del software AWS IoT Greengrass principal dentro del contenedor de Docker. Este parámetro solo es necesario si ha creado un archivo de entorno para establecer variables de entorno. Si no creó un archivo de entorno, puede usar el parámetro de entorno para configurar las variables directamente en su archivo Compose.

> **Note**
>
> Para ejecutar su contenedor de Docker con mayor seguridad, puede usar `cap_drop` y `cap_add` en su archivo Compose para habilitar de forma selectiva las capacidades de Linux para su contenedor. Para obtener más información, consulte Privilegios en tiempo de ejecución y capacidades de Linux en el sitio web de Docker.

3. Para iniciar el contenedor, ejecute el siguiente comando.

```
docker-compose -f docker-compose.yml up
```

Siguientes pasos

AWS IoT Greengrass El software principal se ejecuta ahora en un contenedor de Docker. Ejecute el siguiente comando para recuperar el ID del contenedor en el contenedor de ejecución actual.

```
docker ps
```

A continuación, puede ejecutar el siguiente comando para acceder al contenedor y explorar el software AWS IoT Greengrass principal que se ejecuta dentro del contenedor.

```
docker exec -it container-id /bin/bash
```

Para obtener más información sobre cómo crear un componente simple, consulte Paso 4: Desarrollo y prueba de un componente en su dispositivo en Tutorial: Introducción a AWS IoT Greengrass V2

> **Note**
>
> Cuando utiliza `docker exec` para ejecutar comandos dentro del contenedor de Docker, esos comandos no se capturan en los registros de Docker. Para registrar los comandos

en los registros de Docker, asocie un intérprete de comandos interactivo al contenedor de Docker. Para obtener más información, consulte Asociación de un intérprete de comandos interactivo a un contenedor de Docker.

El archivo de registro AWS IoT Greengrass Core se llama `greengrass.log` y se encuentra en`/greengrass/v2/logs`. Los archivos de registro de componentes también se encuentran en el mismo directorio. Para copiar los registros de Greengrass en un directorio temporal del host, ejecute el siguiente comando:

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Si desea conservar los registros después de que un contenedor se cierre o se haya eliminado, le recomendamos que monte únicamente el directorio `/greengrass/v2/logs` en el directorio de registros temporales del host, en lugar de montar todo el directorio de Greengrass. Para obtener más información, consulte Conservación de los registros de Greengrass fuera del contenedor de Docker.

Para detener un contenedor AWS IoT Greengrass Docker en ejecución, ejecute `docker stop` o`docker-compose -f docker-compose.yml stop`. Esta acción envía `SIGTERM` al proceso de Greengrass y cierra todos los procesos asociados que se iniciaron en el contenedor. El contenedor de Docker se inicializa con el ejecutable `docker-init` como PID 1, lo que ayuda a eliminar los procesos zombis restantes. Para obtener más información, consulte la sección Especificar un proceso de instalación de la documentación de Docker.

Para obtener más información acerca de la solución de problemas relacionados con la ejecución de AWS IoT Greengrass en un contenedor de Docker, consulte Solución de problemas AWS IoT Greengrass en un contenedor Docker.

Solución de problemas AWS IoT Greengrass en un contenedor Docker

Usa la siguiente información como ayuda para solucionar problemas relacionados con la ejecución AWS IoT Greengrass en un contenedor de Docker y para depurar los problemas AWS IoT Greengrass en el contenedor de Docker.

Temas

- Solución de problemas al ejecutar el contenedor de Docker
- Depuración en un contenedor de Docker AWS IoT Greengrass

Solución de problemas al ejecutar el contenedor de Docker

Usa la siguiente información para solucionar problemas relacionados con la ejecución AWS IoT Greengrass en un contenedor de Docker.

Temas

- Error: No se puede realizar un inicio de sesión interactivo desde un dispositivo que no sea TTY
- Error: opciones desconocidas: - no-include-email
- Error: A firewall is blocking file Sharing between windows and the containers.
- Error: se produjo un error (AccessDeniedException) al llamar a la GetAuthorizationToken operación: el usuario: arn:aws:iam: ::user/ <user-name>no account-id está autorizado a realizar: ecr: on resource: * GetAuthorizationToken
- Error: Ha alcanzado su límite de tasa de cambios

Error: No se puede realizar un inicio de sesión interactivo desde un dispositivo que no sea TTY

Este error puede producirse al ejecutar el comando `aws ecr get-login-password`. Asegúrese de haber instalado la AWS CLI versión 2 o 1 más reciente. Le recomendamos que utilice la AWS CLI versión 2. Para obtener más información, consulte Installing the AWS CLI en la Guía del usuario de AWS Command Line Interface .

Error: opciones desconocidas: - no-include-email

Este error puede producirse al ejecutar el comando `aws ecr get-login`. Asegúrese de tener instalada la última AWS CLI versión (por ejemplo, Run:`pip install awscli --upgrade --user`). Para obtener más información, consulte Instalación de AWS Command Line Interface en Microsoft Windows en la Guía del AWS Command Line Interface usuario.

Error: A firewall is blocking file Sharing between windows and the containers.

Puede que reciba este error o un mensaje `Firewall Detected` al ejecutar Docker en un equipo Windows. Esto también puede ocurrir si ha iniciado sesión en una red privada virtual (VPN) y su configuración de red impide el montaje de la unidad compartida. En esta situación, desactive la VPN y vuelva a ejecutar el contenedor Docker.

Error: se produjo un error (AccessDeniedException) al llamar a la GetAuthorizationToken operación: el usuario: arn:aws:iam: ::user/ <user-name>no ***account-id*** está autorizado a realizar: ecr: on resource: * GetAuthorizationToken

Puede recibir este error al ejecutar el comando `aws ecr get-login-password` si no tiene los permisos suficientes para acceder a un repositorio de Amazon ECR. Para obtener más información, consulte los Ejemplos de políticas de repositorios de Amazon ECR y el Acceso a un repositorio de Amazon ECR en la Guía del usuario de Amazon ECR.

Error: Ha alcanzado su límite de tasa de cambios

Docker Hub limita la cantidad de solicitudes de cambio que pueden realizar los usuarios anónimos y gratuitos de Docker Hub. Si supera los límites de tasa de solicitudes de cambio de usuarios anónimos o gratuitos, entonces recibe uno de los siguientes errores:

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

Para resolver estos errores, puede esperar unas horas antes de intentar realizar otra solicitud de cambio. Si planea enviar una gran cantidad de solicitudes de cambio de forma constante, visite el sitio web de Docker Hub para obtener información sobre los límites de tasa y las opciones para autenticar y actualizar su cuenta de Docker.

Depuración en un contenedor de Docker AWS IoT Greengrass

Para depurar problemas con un contenedor de Docker, puede conservar los registros del tiempo de ejecución de Greengrass o asociar un intérprete de comandos interactivo al contenedor de Docker.

Conservación de los registros de Greengrass fuera del contenedor de Docker

Tras detener un AWS IoT Greengrass contenedor, puede usar el siguiente `docker cp` comando para copiar los registros de Greengrass del contenedor de Docker a un directorio de registros temporal.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Para conservar los registros incluso después de que un contenedor salga o se elimine, debe ejecutar el contenedor AWS IoT Greengrass Docker después de montar el directorio en un enlace. */greengrass/v2*/logs

Para montar el */greengrass/v2*/logs directorio de forma binaria, realiza una de las siguientes acciones cuando ejecutes un nuevo contenedor de Docker. AWS IoT Greengrass

- Incluya -v */tmp/logs*:*/greengrass/v2*/logs:ro en su comando docker run.

  Modifique el bloque volumes en el archivo de Compose para incluir la siguiente línea antes de ejecutar el comando docker-compose up.

```
volumes:
 - /tmp/logs:/greengrass/v2/logs:ro
```

A continuación, puede comprobar sus registros */tmp/logs* en su host para ver los registros de Greengrass mientras AWS IoT Greengrass se ejecuta dentro del contenedor de Docker.

Para obtener información sobre cómo ejecutar contenedores de Docker de Greengrass, consulte Se ejecuta AWS IoT Greengrass en Docker con aprovisionamiento manual y Se ejecuta AWS IoT Greengrass en Docker con aprovisionamiento automático.

Asociación de un intérprete de comandos interactivo a un contenedor de Docker

Cuando utiliza docker exec para ejecutar comandos dentro del contenedor de Docker, esos comandos no se capturan en los registros de Docker. Registrar los comandos en los registros de Docker puede ayudarlo a investigar el estado del contenedor de Docker de Greengrass. Realice una de las siguientes acciones:

- Ejecute el siguiente comando en un terminal independiente para adjuntar la entrada, la salida y el error estándar del terminal al contenedor en ejecución. Esto le permite ver y controlar el contenedor de Docker desde su terminal actual.

```
docker attach container-id
```

- Ejecute el siguiente comando en un terminal independiente. Esto le permite ejecutar sus comandos en modo interactivo, incluso si el contenedor no está conectado.

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

Para obtener información general sobre la AWS IoT Greengrass solución de problemas, consulte. Resolución de problemas

Configurar el software AWS IoT Greengrass principal

El software AWS IoT Greengrass Core ofrece opciones que puede utilizar para configurar el software. Puede crear implementaciones para configurar el software AWS IoT Greengrass principal en cada dispositivo principal.

Temas

- Implementación del componente núcleo de Greengrass
- Configuración del núcleo de Greengrass como un servicio del sistema
- Control de la asignación de memoria con las opciones de JVM
- Configuración del usuario que ejecuta los componentes
- Configuración de los límites de recursos del sistema para los componentes
- Realizar la conexión en el puerto 443 o a través de un proxy de red
- Use un certificado de dispositivo firmado por una CA privada
- Configuración de los tiempos de espera y los ajustes de caché de MQTT
- Configurar Greengrass Nucleus en la red IPv6

Implementación del componente núcleo de Greengrass

AWS IoT Greengrass proporciona el software AWS IoT Greengrass Core como un componente que puede implementar en sus dispositivos principales de Greengrass. Puede crear una implementación para aplicar la misma configuración a varios dispositivos principales de Greengrass. Para obtener más información, consulte Núcleo de Greengrass y Actualización del software AWS IoT Greengrass Core (OTA).

Configuración del núcleo de Greengrass como un servicio del sistema

Debe configurar el software AWS IoT Greengrass Core como un servicio del sistema en el sistema de inicio de su dispositivo para hacer lo siguiente:

- Inicie el software AWS IoT Greengrass Core cuando se inicie el dispositivo. Esta es una buena práctica si administra grandes flotas de dispositivos.

- Instale y ejecute los componentes del complemento. Varios AWS de los componentes proporcionados son componentes de complementos, lo que les permite interactuar directamente con el núcleo de Greengrass. Para obtener más información acerca de los tipos de componentes, consulte Tipos de componentes.
- Aplique las actualizaciones over-the-air (OTA) al software principal del dispositivo AWS IoT Greengrass principal. Para obtener más información, consulte Actualización del software AWS IoT Greengrass Core (OTA).
- Permita que los componentes reinicien el software AWS IoT Greengrass principal o el dispositivo principal cuando una implementación actualice el componente a una nueva versión o actualice ciertos parámetros de configuración. Para obtener más información, consulte el paso del ciclo de vida de arranque.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Temas

- Configuración del núcleo como un servicio del sistema (Linux)
- Configuración del núcleo como un servicio del sistema (Windows)

Configuración del núcleo como un servicio del sistema (Linux)

Los dispositivos Linux admiten diferentes sistemas de inicio, como initd, systemd y SystemV. El `--setup-system-service true` argumento se utiliza al instalar el software AWS IoT Greengrass Core para iniciar el núcleo como un servicio del sistema y configurarlo para que se inicie cuando se inicie el dispositivo. El instalador configura el software AWS IoT Greengrass Core como un servicio del sistema con systemd.

También puede configurar manualmente el núcleo para que se ejecute como un servicio del sistema. En el siguiente ejemplo se muestra un archivo de servicio para systemd.

```
[Unit]
Description=Greengrass Core
```

```
[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

[Install]
WantedBy=multi-user.target
```

Tras configurar el servicio del sistema, puede ejecutar los siguientes comandos para configurar el inicio del dispositivo al arrancar y para iniciar o detener el software AWS IoT Greengrass principal.

- Comprobar el estado del servicio (systemd)

```
sudo systemctl status greengrass.service
```

- Permitir que el núcleo se inicie al arrancar el dispositivo.

```
sudo systemctl enable greengrass.service
```

- Impedir que el núcleo se inicie al arrancar el dispositivo.

```
sudo systemctl disable greengrass.service
```

- Para iniciar el software AWS IoT Greengrass principal.

```
sudo systemctl start greengrass.service
```

- Para detener el software AWS IoT Greengrass Core.

```
sudo systemctl stop greengrass.service
```

Configuración del núcleo como un servicio del sistema (Windows)

El `--setup-system-service true` argumento se utiliza al instalar el software AWS IoT Greengrass Core para iniciar el núcleo como un servicio de Windows y configurarlo para que se inicie al arrancar el dispositivo.

Tras configurar el servicio, puede ejecutar los siguientes comandos para configurar el inicio del dispositivo durante el arranque y para iniciar o detener el software AWS IoT Greengrass Core. Debe ejecutar la línea de comandos o PowerShell como administrador para ejecutar estos comandos.

Windows Command Prompt (CMD)

- Comprobar el estado del servicio

```
sc query "greengrass"
```

- Permitir que el núcleo se inicie al arrancar el dispositivo.

```
sc config "greengrass" start=auto
```

- Impedir que el núcleo se inicie al arrancar el dispositivo.

```
sc config "greengrass" start=disabled
```

- Para iniciar el software AWS IoT Greengrass principal.

```
sc start "greengrass"
```

- Para detener el software AWS IoT Greengrass Core.

```
sc stop "greengrass"
```

> **Note**
>
> En los dispositivos Windows, el software AWS IoT Greengrass Core ignora esta señal de apagado mientras cierra los procesos de los componentes de Greengrass. Si el software AWS IoT Greengrass Core ignora la señal de apagado al ejecutar este comando, espere unos segundos e inténtelo de nuevo.

PowerShell

- Comprobar el estado del servicio

```
Get-Service -Name "greengrass"
```

- Permitir que el núcleo se inicie al arrancar el dispositivo.

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- Impedir que el núcleo se inicie al arrancar el dispositivo.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- Para iniciar el software AWS IoT Greengrass Core.

```
Start-Service -Name "greengrass"
```

- Para detener el software AWS IoT Greengrass Core.

```
Stop-Service -Name "greengrass"
```

> **Note**
>
> En los dispositivos Windows, el software AWS IoT Greengrass Core ignora esta señal de apagado mientras cierra los procesos de los componentes de Greengrass. Si el software AWS IoT Greengrass Core ignora la señal de apagado al ejecutar este comando, espere unos segundos e inténtelo de nuevo.

Control de la asignación de memoria con las opciones de JVM

Si utilizas un dispositivo con memoria limitada, puedes usar las opciones de la máquina virtual Java (JVM) para controlar el tamaño máximo del montón, los modos de recolección de basura y las opciones del compilador, que controlan la cantidad de memoria que AWS IoT Greengrass utiliza el software Core. AWS IoT Greengrass El tamaño del montón de la JVM determina la cantidad de memoria que puede usar una aplicación antes de que se produzca la recopilación de elementos no utilizados o antes de que la aplicación se quede sin memoria. El tamaño de montón máximo especifica la cantidad máxima de memoria que la JVM puede asignar al ampliar el montón durante un periodo de actividad intensa.

Para controlar la asignación de memoria, cree una nueva implementación o revise una implementación existente que incluya el componente núcleo y especifique las opciones de JVM en el parámetro de configuración `jvmOptions` de la configuración del componente núcleo.

En función de sus necesidades, puede ejecutar el software AWS IoT Greengrass Core con una asignación de memoria reducida o con una asignación de memoria mínima.

Asignación de memoria reducida

Para ejecutar el software AWS IoT Greengrass Core con una asignación de memoria reducida, le recomendamos que utilice el siguiente ejemplo de actualización de combinación de configuraciones para configurar las opciones de JVM en su configuración de núcleo:

```
{
  "jvmOptions": "-XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

Asignación de memoria mínima

Para ejecutar el software AWS IoT Greengrass Core con una asignación de memoria mínima, le recomendamos que utilice el siguiente ejemplo de actualización de combinación de configuraciones para configurar las opciones de JVM en su configuración de núcleo:

```
{
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
}
```

> **Important**
>
> La ejecución AWS IoT Greengrass del software Core con una asignación de memoria mínima puede tener un impacto significativo en el rendimiento en sistemas de baja especificación, ya que la JVM procesará más cuando utilice menos memoria. Recomendamos ajustar las opciones para equilibrar las necesidades de memoria y rendimiento.

Estos ejemplos de actualizaciones de combinación de configuraciones usan las siguientes opciones de JVM:

`-XX:+UseSerialGC`

Especifica el uso de la recopilación de elementos no utilizados en serie para el espacio del montón de JVM. La recopilación de elementos no utilizados en serie es más lenta, pero usa menos memoria que otras implementaciones de recopilación de elementos no utilizados de JVM.

`-XX:TieredStopAtLevel=1`

Indica a la JVM que utilice el compilador de Java just-in-time (JIT) una vez. Como el código compilado JIT ocupa espacio en la memoria del dispositivo, usar el compilador JIT más de una vez consume más memoria que una sola compilación.

`-Xmx`*NN*`m`

Establece el tamaño máximo de los montones de JVM.

> **Important**
>
> Si el tamaño máximo del montón es demasiado bajo, se pueden producir errores o ralentizar el rendimiento. out-of-memory Recomendamos medir el uso actual del montón antes de establecer un tamaño máximo con la opción `-Xmx`*NN*`m`. Configure su JVM con la opción de JVM `-XX:NativeMemoryTracking=detail`. A continuación, mida el uso actual del montón mediante la solicitud de comando `VM.native_memory` dentro de la utilidad jcmd.

Si la medición del montón no es una opción, use `-Xmx64m` como valor inicial para limitar el tamaño del montón a 64 MB. A partir de ahí, puede reducir gradualmente el tamaño máximo del montón. Para una asignación de memoria mínima, use `-Xmx32m` como valor inicial para limitar el tamaño del montón a 32 MB.

Puede aumentar o disminuir el valor `-Xmx` en función de sus necesidades reales; sin embargo, le recomendamos encarecidamente que no establezca el tamaño máximo del montón por debajo de 16 MB. La cantidad de tamaño de montón de JVM necesaria también puede variar con el tiempo en función de los componentes del complemento implementados en el dispositivo principal. Si el tamaño máximo del montón es demasiado bajo para su entorno, es posible que el software AWS IoT Greengrass Core detecte errores inesperados debido a la falta de memoria. Si el rendimiento es más lento o se producen errores debido a una memoria insuficiente, vuelva a una configuración que se sepa que es correcta. Por ejemplo, si el tamaño normal del montón asignado es `41428KB`, use `-Xmx40m` para limitar ligeramente el uso del montón.

`-Xint`

Indica a la JVM que no utilice el compilador just-in-time (JIT). En su lugar, la JVM se ejecuta en modo de solo interpretación. Este modo es más lento (potencialmente 20 veces más lento para

las implementaciones en sistemas de gama baja) que ejecutar código compilado por JIT; sin embargo, el código compilado no ocupa espacio en la memoria.

Para obtener información sobre la creación de actualizaciones de combinación de configuraciones, consulte Actualización de las configuraciones de los componentes.

Configuración del usuario que ejecuta los componentes

El software AWS IoT Greengrass Core puede ejecutar procesos componentes como usuario y grupo del sistema distintos del que ejecuta el software. Esto aumenta la seguridad, ya que puede ejecutar el software AWS IoT Greengrass principal como usuario root o como usuario administrador, sin conceder esos permisos a los componentes que se ejecutan en el dispositivo principal.

En la siguiente tabla se indican los tipos de componentes que el software AWS IoT Greengrass principal puede ejecutar como usuario que especifique. Para obtener más información, consulte Tipos de componentes.

Tipo de componente	Configuración del usuario del componente
Núcleo	 No
Complemento	 No
Genérico	 Sí

Tipo de componente	Configuración del usuario del componente
Lambda (no contenerizado)	 Sí
Lambda (en contenedor)	 Sí

Debe crear el usuario del componente antes de poder especificarlo en una configuración de implementación. En los dispositivos basados en Windows, también debe almacenar el nombre de usuario y la contraseña del usuario en la instancia del administrador de credenciales de la LocalSystem cuenta. Para obtener más información, consulte Configuración de un usuario del componente en dispositivos Windows.

Al configurar el usuario del componente en un dispositivo basado en Linux, también puede especificar un grupo si lo desea. Especifique el usuario y el grupo separados por dos puntos (`:`) con el siguiente formato: *`user`*`:`*`group`*. Si no especifica ningún grupo, el software AWS IoT Greengrass principal utilizará de forma predeterminada el grupo principal del usuario. Puede usar el nombre o el ID para identificar al usuario y al grupo.

En los dispositivos basados en Linux, también puede ejecutar componentes como un usuario del sistema que no existe (también denominado usuario desconocido) para aumentar la seguridad. Un proceso de Linux puede indicar cualquier otro proceso que ejecute el mismo usuario. Un usuario desconocido no ejecuta otros procesos, por lo que puede ejecutar componentes como un usuario desconocido para evitar que los componentes señalen otros componentes del dispositivo principal. Para ejecutar los componentes como un usuario desconocido, especifique un ID de usuario que no exista en el dispositivo principal. También puede especificar un ID de grupo que no existe para que se ejecute como un grupo desconocido.

Puede configurar el usuario para cada componente y para cada dispositivo principal.

- Configuración para un componente

Puede configurar cada componente para que se ejecute con un usuario específico de ese componente. Al crear una implementación, puede especificar el usuario para cada componente de la configuración `runWith` de ese componente. El software AWS IoT Greengrass principal ejecuta los componentes como el usuario especificado si los configura. De lo contrario, se ejecutarán los componentes de forma predeterminada como el usuario predeterminado que configure para el dispositivo principal. Para obtener más información sobre cómo especificar el usuario del componente en la configuración de implementación, consulte el parámetro de configuración `runWith` en Crear implementaciones.

- Configuración del usuario predeterminado para un dispositivo principal

  Puede configurar un usuario predeterminado que el software AWS IoT Greengrass Core utilice para ejecutar los componentes. Cuando el software AWS IoT Greengrass principal ejecuta un componente, comprueba si ha especificado un usuario para ese componente y lo utiliza para ejecutar el componente. Si el componente no especifica un usuario, el software AWS IoT Greengrass principal ejecuta el componente como el usuario predeterminado que configuró para el dispositivo principal. Para obtener más información, consulte Configuración del usuario del componente predeterminado.

> **Note**
>
> En los dispositivos basados en Windows, debe especificar al menos un usuario predeterminado para ejecutar los componentes.
>
> En los dispositivos basados en Linux, se deben tener en cuenta las siguientes consideraciones si no se configura un usuario para que ejecute componentes:
>
> - Si ejecuta el software AWS IoT Greengrass principal como root, el software no ejecutará los componentes. Debe especificar un usuario predeterminado para ejecutar los componentes si ejecuta los componentes como raíz.
> - Si ejecuta el software AWS IoT Greengrass Core como un usuario que no es root, el software ejecuta los componentes como ese usuario.

Temas

- Configuración de un usuario del componente en dispositivos Windows
- Configuración del usuario del componente predeterminado

Configuración de un usuario del componente en dispositivos Windows

Cómo configurar un usuario del componente en un dispositivo basado en Windows

1. Cree el usuario del componente en la LocalSystem cuenta del dispositivo.

   ```
   net user /add component-user password
   ```

2. Utilice la PsExec utilidad de Microsoft para almacenar el nombre de usuario y la contraseña del usuario del componente en la instancia de Credential Manager de la LocalSystem cuenta.

   ```
   psexec -s cmd /c cmdkey /generic:component-user /user:component-user /pass:password
   ```

   > **Note**
   >
   > En los dispositivos basados en Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe utilizar PsExec la utilidad para almacenar la información del usuario del componente en LocalSystem la cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la cuenta. LocalSystem

Configuración del usuario del componente predeterminado

Puede usar una implementación para configurar el usuario predeterminado en un dispositivo principal. En esta implementación, se actualiza la configuración de los componentes del núcleo.

> **Note**
>
> También puede configurar el usuario predeterminado al instalar el software AWS IoT Greengrass principal con la `--component-default-user` opción. Para obtener más información, consulte Instalación del software AWS IoT Greengrass Core.

Cree una implementación que especifique la siguiente actualización de configuración para el componente `aws.greengrass.Nucleus`.

Linux

```
{
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

> **Note**
>
> El usuario que especifique debe existir y el nombre de usuario y la contraseña de este usuario deben estar almacenados en la instancia del administrador de credenciales de la LocalSystem cuenta de su dispositivo Windows. Para obtener más información, consulte Configuración de un usuario del componente en dispositivos Windows.

En el siguiente ejemplo, se define una implementación para un dispositivo basado en Linux que configura `ggc_user` como usuario predeterminado y `ggc_group` como grupo predeterminado. La actualización de configuración `merge` requiere un objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}
```

Configuración de los límites de recursos del sistema para los componentes

> **Note**
>
> Esta función está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente pueden usar en el dispositivo principal.

En la siguiente tabla, se muestran los tipos de componentes que admiten los límites de recursos del sistema. Para obtener más información, consulte Tipos de componentes.

Tipo de componente	Configuración de los límites de recursos del sistema
Núcleo	 No
Complemento	 No
Genérico	 Sí
Lambda (no contenerizado)	 Sí

Tipo de componente	Configuración de los límites de recursos del sistema
Lambda (en contenedor)	 No

> **Important**
>
> Los límites de recursos del sistema no se admiten cuando se ejecuta el software AWS IoT Greengrass principal en un contenedor de Docker.

Puede configurar los límites de recursos del sistema para cada componente y para cada dispositivo principal.

- Configuración para un componente

  Puede configurar cada componente con los límites de recursos del sistema específicos para ese componente. Al crear una implementación, puede especificar los límites de recursos del sistema para cada componente de la implementación. Si el componente admite los límites de recursos del sistema, el software AWS IoT Greengrass Core aplica los límites a los procesos del componente. Si no especificas los límites de recursos del sistema para un componente, el software AWS IoT Greengrass principal utilizará los valores predeterminados que hayas configurado para el dispositivo principal. Para obtener más información, consulte Crear implementaciones.

- Configuración de los valores predeterminados para un dispositivo principal

  Puede configurar los límites de recursos del sistema predeterminados que el software AWS IoT Greengrass Core aplica a los componentes que admiten estos límites. Cuando el software AWS IoT Greengrass principal ejecuta un componente, aplica los límites de recursos del sistema que especifique para ese componente. Si ese componente no especifica los límites de recursos del sistema, el software AWS IoT Greengrass Core aplica los límites de recursos del sistema predeterminados que usted configure para el dispositivo principal. Si no especificas los límites de recursos del sistema predeterminados, el software AWS IoT Greengrass Core no aplicará ningún

límite de recursos del sistema de forma predeterminada. Para obtener más información, consulte Configuración de los límites de recursos del sistema predeterminados.

Configuración de los límites de recursos del sistema predeterminados

Puede implementar el componente núcleo de Greengrass para configurar los límites de recursos del sistema predeterminados para un dispositivo principal. Para configurar los límites de recursos del sistema predeterminados, cree una implementación que especifique la siguiente actualización de configuración para el componente `aws.greengrass.Nucleus`.

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

En el siguiente ejemplo, se define una implementación que configura el límite de tiempo de la CPU en 2, lo que equivale al 50 % de uso en un dispositivo con 4 núcleos de CPU. Este ejemplo también configura el uso de memoria en 100 MB.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":\"cpus\":2,\"memory
\":102400}}}"
      }
    }
  }
}
```

Realizar la conexión en el puerto 443 o a través de un proxy de red

AWS IoT Greengrass los dispositivos principales se comunican AWS IoT Core mediante el protocolo de mensajería MQTT con autenticación de cliente TLS. Convencionalmente, MQTT sobre TLS utiliza el puerto 8883. Sin embargo, como medida de seguridad, los entornos restrictivos podrían

limitar el tráfico de entrada y salida a un pequeño rango de puertos TCP. Por ejemplo, el firewall de una compañía podría abrir el puerto 443 para el tráfico HTTPS, pero cerrar otros puertos que se utilizan para protocolos menos frecuentes, como, por ejemplo, el puerto 8883 para tráfico MQTT. Otros entornos restrictivos podrían requerir que todo el tráfico pase a través de un proxy antes de conectarse a Internet.

> **Note**
>
> Los dispositivos principales de Greengrass que ejecutan el componente núcleo de Greengrass v2.0.3 y versiones anteriores utilizan el puerto 8443 para conectarse al punto final del plano de datos. AWS IoT Greengrass Estos dispositivos deben poder conectarse a este punto de conexión en el puerto 8443. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

Para habilitar la comunicación en estos escenarios, AWS IoT Greengrass proporciona las siguientes opciones de configuración:

- Comunicación MQTT a través del puerto 443. Si su red permite realizar conexiones al puerto 443, puede configurar el dispositivo principal de Greengrass para usar el puerto 443 para tráfico MQTT en lugar del puerto predeterminado 8883. Esto puede ser una conexión directa al puerto 443 o una conexión a través de un servidor proxy de red. A diferencia de la configuración predeterminada, que usa la autenticación de cliente basada en certificados, el MQTT del puerto 443 usa la función de rol de servicio del dispositivo para la autenticación.

  Para obtener más información, consulte Configuración de MQTT a través del puerto 443.
- Comunicación HTTPS a través del puerto 443. El software AWS IoT Greengrass principal envía el tráfico HTTPS a través del puerto 8443 de forma predeterminada, pero puede configurarlo para que utilice el puerto 443. AWS IoT Greengrass utiliza la extensión TLS de Application Layer Protocol Network (ALPN) para habilitar esta conexión. Como en el caso de la configuración predeterminada, HTTPS en el puerto 443 usa la autenticación de cliente basada en certificados.

  > **Important**
  >
  > Para usar ALPN y habilitar la comunicación HTTPS a través del puerto 443, el dispositivo principal debe ejecutar la actualización 252 o posterior de Java 8. Todas las actualizaciones de la versión 9 y posteriores de Java también son compatibles con ALPN.

Para obtener más información, consulte Configuración de HTTPS a través del puerto 443.

- Conexión a través de un proxy de red. Puede configurar un servidor proxy de red para que actúe como intermediario para conectarse al dispositivo principal de Greengrass. AWS IoT Greengrass admite la autenticación básica para los proxy de HTTP y HTTPS.

  Los dispositivos principales de Greengrass deben ejecutar la versión 2.5.0 o versiones posteriores del núcleo de Greengrass para usar proxies HTTPS.

  El software AWS IoT Greengrass principal transfiere la configuración del proxy a los componentes a través de las `ALL_PROXY` variables de `NO_PROXY` entorno `HTTP_PROXY``HTTPS_PROXY`,, y. Los componentes deben utilizar estos ajustes para conectarse a través del proxy. Los componentes usan bibliotecas (como boto3 o cURL y los paquetes `requests` de Python) que suelen usar estas variables de entorno de forma predeterminada para hacer conexiones. Si un componente también especifica estas variables de entorno, AWS IoT Greengrass no las invalida.

  Para obtener más información, consulte Configuración de un proxy de red.

Configuración de MQTT a través del puerto 443

Puede configurar MQTT a través del puerto 443 en los dispositivos principales existentes o al instalar el software AWS IoT Greengrass Core en un dispositivo principal nuevo.

Temas

- Configuración de MQTT a través del puerto 443 en los dispositivos principales existentes
- Configuración de MQTT a través del puerto 443 durante la instalación

Configuración de MQTT a través del puerto 443 en los dispositivos principales existentes

Puede usar una implementación para configurar MQTT a través del puerto 443 en un único dispositivo principal o en un grupo de dispositivos principales. En esta implementación, se actualiza la configuración de los componentes del núcleo. El núcleo se reinicia al actualizar su configuración `mqtt`.

Para configurar MQTT a través del puerto 443, cree una implementación que especifique la siguiente actualización de configuración para el componente `aws.greengrass.Nucleus`.

```
{
```

```
  "mqtt": {
    "port": 443
  }
}
```

En el siguiente ejemplo, se define una implementación que configura MQTT a través del puerto 443. La actualización de configuración `merge` requiere un objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

Configuración de MQTT a través del puerto 443 durante la instalación

Puede configurar MQTT a través del puerto 443 al instalar el software AWS IoT Greengrass Core en un dispositivo principal. Use el argumento del instalador `--init-config` para configurar MQTT a través del puerto 443. Puede especificar este argumento al realizar la instalación con aprovisionamiento manual, aprovisionamiento de flota o aprovisionamiento personalizado.

Configuración de HTTPS a través del puerto 443

Esta característica requiere la versión 2.0.4 o posterior de Núcleo de Greengrass.

Puede configurar HTTPS a través del puerto 443 en los dispositivos principales existentes o al instalar el software AWS IoT Greengrass Core en un dispositivo principal nuevo.

Temas

- Configuración de HTTPS a través del puerto 443 en los dispositivos principales existentes
- Configuración de HTTPS a través del puerto 443 durante la instalación

Configuración de HTTPS a través del puerto 443 en los dispositivos principales existentes

Puede usar una implementación para configurar HTTPS a través del puerto 443 en un único dispositivo principal o en un grupo de dispositivos principales. En esta implementación, se actualiza la configuración de los componentes del núcleo.

Para configurar HTTPS a través del puerto 443, cree una implementación que especifique la siguiente actualización de configuración para el componente `aws.greengrass.Nucleus`.

```
{
  "greengrassDataPlanePort": 443
}
```

En el ejemplo siguiente, se define una implementación que configura HTTPS a través del puerto 443. La actualización de configuración `merge` requiere un objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

Configuración de HTTPS a través del puerto 443 durante la instalación

Puede configurar HTTPS a través del puerto 443 al instalar el software AWS IoT Greengrass Core en un dispositivo principal. Use el argumento del instalador `--init-config` para configurar HTTPS a través del puerto 443. Puede especificar este argumento al realizar la instalación con aprovisionamiento manual, aprovisionamiento de flota o aprovisionamiento personalizado.

Configuración de un proxy de red

Siga el procedimiento de esta sección para configurar los dispositivos principales de Greengrass para que se conecten a Internet a través de un proxy de red HTTP o HTTPS. Para obtener más información acerca de los puntos de conexión y los puertos que usan los dispositivos principales, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

Important

Si su dispositivo principal ejecuta una versión del núcleo de Greengrass anterior a la 2.4.0, la función de su dispositivo debe permitir los siguientes permisos para usar un proxy de red:

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

Esto es necesario porque el dispositivo utiliza AWS las credenciales del servicio de intercambio de fichas para autenticar las conexiones MQTT. AWS IoT El dispositivo utiliza MQTT para recibir e instalar las implementaciones desde allí Nube de AWS, por lo que el dispositivo no funcionará a menos que defina estos permisos en función de su función. Los dispositivos suelen usar certificados X.509 para autenticar las conexiones MQTT, pero los dispositivos no pueden hacerlo para autenticarse cuando usan un proxy.
Para obtener más información acerca de cómo configurar el rol de dispositivo, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Temas

- Configuración de un proxy de red en dispositivos principales existentes
- Configuración de un proxy de red durante la instalación
- Permita que el dispositivo principal confíe en un proxy HTTPS
- El objeto networkProxy

Configuración de un proxy de red en dispositivos principales existentes

Puede usar una implementación para configurar un proxy de red en un único dispositivo principal o en un grupo de dispositivos principales. En esta implementación, se actualiza la configuración de los componentes del núcleo. El núcleo se reinicia al actualizar su configuración `networkProxy`.

Para configurar un proxy de red, cree una implementación para el componente `aws.greengrass.Nucleus` que combine la siguiente actualización de configuración. Esta actualización de configuración contiene el objeto networkProxy.

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

En el ejemplo siguiente, se define una implementación que configura un proxy de red. La actualización de configuración `merge` requiere un objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
}
```

Configuración de un proxy de red durante la instalación

Puede configurar un proxy de red al instalar el software AWS IoT Greengrass Core en un dispositivo principal. Use el argumento del instalador `--init-config` para configurar el proxy de red. Puede especificar este argumento al realizar la instalación con aprovisionamiento manual, aprovisionamiento de flota o aprovisionamiento personalizado.

Permita que el dispositivo principal confíe en un proxy HTTPS

Al configurar un dispositivo principal para que use un proxy HTTPS, debe agregar la cadena de certificados del servidor proxy a la del dispositivo principal para que pueda confiar en el proxy HTTPS. De lo contrario, el dispositivo principal podría encontrar errores al intentar enrutar el tráfico a través del proxy. Agregue el certificado de CA del servidor proxy al archivo de certificado de la CA raíz de Amazon en el dispositivo principal.

Habilitación del dispositivo principal para que confíe en el proxy HTTPS

1. Encuentre el archivo de certificado de la CA raíz de Amazon en el dispositivo principal.

   - Si instaló el software AWS IoT Greengrass Core con aprovisionamiento automático, el archivo de certificado de CA raíz de Amazon existe en */greengrass/v2*/rootCA.pem.
   - Si instaló el software AWS IoT Greengrass Core con aprovisionamiento manual o de flota, es posible que el archivo de certificado CA raíz de Amazon esté en */greengrass/v2*/AmazonRootCA1.pem.

   Si el certificado de la CA raíz de Amazon no existe en estas ubicaciones, registre la propiedad `system.rootCaPath` en */greengrass/v2*/config/effectiveConfig.yaml para encontrar su ubicación.

2. Agregue el contenido del archivo de certificado de la CA del servidor proxy al archivo de certificado de la CA raíz de Amazon.

   En el ejemplo siguiente, se muestra un certificado de la CA del servidor proxy agregado al archivo de certificado de la CA raíz de Amazon.

```
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

El objeto networkProxy

Utilice el objeto `networkProxy` para especificar información sobre el proxy de red. Este objeto contiene la siguiente información:

`noProxyAddresses`

(Opcional) Una lista separada por comas de direcciones IP o nombres de host que están exentos del proxy.

`proxy`

El proxy al que conectar. Este objeto contiene la siguiente información:

`url`

La dirección URL del servidor proxy, en el formato `scheme://userinfo@host:port`.

- `scheme`: el esquema, que debe ser `http` o `https`.

  > **Important**
  >
  > Los dispositivos principales de Greengrass deben ejecutar la versión 2.5.0 o versiones posteriores del núcleo de Greengrass para usar proxies HTTPS. Si configura un proxy HTTPS, debe agregar el certificado de la CA del servidor proxy al certificado de la CA raíz de Amazon del dispositivo principal. Para obtener más información, consulte Permita que el dispositivo principal confíe en un proxy HTTPS.

- `userinfo`: (opcional) la información de nombre de usuario y contraseña. Si especifica esta información en `url`, el dispositivo principal de Greengrass ignora los campos `username` y `password`.
- `host`: el nombre de host o dirección IP del servidor proxy.
- `port`: (opcional) el número de puerto. Si no especifica el puerto, el dispositivo principal de Greengrass usará los siguientes valores predeterminados:
  - `http`: 80
  - `https`: 443

`username`

(Opcional) El nombre de usuario que autentica el servidor proxy.

`password`

(Opcional) La contraseña para autenticarse en el servidor proxy.

Use un certificado de dispositivo firmado por una CA privada

Si usa una autoridad de certificación (CA) privada personalizada, deberá establecer el **`greengrassDataPlaneEndpoint`** del núcleo de Greengrass en **`iotdata`**. Puede configurar esta opción durante la implementación o la instalación mediante el argumento del instalador **`--init-config`**.

Puede personalizar el punto de conexión del plano de datos de Greengrass al que se conecta el dispositivo. Puede establecer esta opción de configuración en **`iotdata`** para que el punto de conexión del plano de datos de Greengrass esté en el mismo punto de conexión que el punto de conexión de datos de IoT, que puede especificar con **`iotDataEndpoint`**.

Configuración de los tiempos de espera y los ajustes de caché de MQTT

En el AWS IoT Greengrass entorno, los componentes pueden utilizar MQTT para comunicarse con ellos. AWS IoT Core El software AWS IoT Greengrass Core gestiona los mensajes MQTT de los componentes. Cuando el dispositivo principal pierde la conexión con la Nube de AWS, el software almacena en caché los mensajes MQTT para volver a intentarlo más adelante cuando se restablezca la conexión. Puede configurar ajustes como los tiempos de espera de los mensajes y el tamaño de la memoria caché. Para obtener más información, consulte `mqtt` y los parámetros de configuración `mqtt.spooler` del componente núcleo de Greengrass.

AWS IoT Core impone cuotas de servicio a su agente de mensajes MQTT. Estas cuotas pueden aplicarse a los mensajes que envíe entre los dispositivos principales y AWS IoT Core. Para obtener más información, consulte Service Quotas al agente de mensajes de AWS IoT Core en Referencia general de AWS.

Configurar Greengrass Nucleus en la red IPv6

Greengrass Nucleus habla a través de AWS IoT Core Greengrass. APIs APIs Compatibilidad con Greengrass en un entorno de IPv6 doble pila.

Para habilitar los puntos finales de doble pila para: IPv6

- Agregue las propiedades del sistema `aws.useDualstackEndpoint=true` y `java.net.preferIPv6Addresses=true` a `jvmOptions`

- Establezca `s3EndpointType` en `DUALSTACK`

Establezca esta opción durante la implementación o aprovisiónela de forma manual con el argumento de instalador `--init-config`. Consulte Using Amazon S3 dual-stack endpoints.

Example código de implementación:

```
{
    "jvmOptions": "-Daws.useDualstackEndpoint=true",
    "s3EndpointType":"DUALSTACK"
}
```

Example **`config.yaml`** mediante aprovisionamiento manual:

```
---
system:
  ...
services:
  aws.greengrass.Nucleus:
    ...
    configuration:
      ...
      jvmOptions: "-Daws.useDualstackEndpoint=true -Djava.net.preferIPv6Addresses=true"
      s3EndpointType: "DUALSTACK"
```

Actualización del software AWS IoT Greengrass Core (OTA)

El software AWS IoT Greengrass Core incluye el componente de núcleo de Greengrass y otros componentes opcionales que puede implementar en sus dispositivos para realizar actualizaciones vía inalámbrica (OTA) del software. Esta característica está integrada en el software AWS IoT Greengrass Core.

Las actualizaciones OTA hacen que sea más eficiente:

- Corrige vulnerabilidades de seguridad.
- Solucionar problemas de estabilidad del software.
- Implementar características nuevas o mejoradas.

Temas

- Requisitos
- Consideraciones para los dispositivos principales
- Comportamiento de actualización del núcleo de Greengrass
- Cómo realizar actualizaciones OTA

Requisitos

Los siguientes requisitos se aplican para la implementación de las actualizaciones OTA del software AWS IoT Greengrass Core:

- El dispositivo principal de Greengrass debe tener una conexión con la Nube de AWS para recibir la implementación.
- El dispositivo principal de Greengrass debe configurarse y aprovisionarse correctamente con certificados y claves para la autenticación con AWS IoT Core y AWS IoT Greengrass.
- El software AWS IoT Greengrass Core debe configurarse y ejecutarse como un servicio del sistema. Las actualizaciones OTA no funcionan si se ejecuta el núcleo desde el archivo JAR, `Greengrass.jar`. Para obtener más información, consulte Configuración del núcleo de Greengrass como un servicio del sistema.

Consideraciones para los dispositivos principales

Antes de realizar una actualización OTA, tenga en cuenta el impacto en los dispositivos principales que actualice y en los dispositivos de cliente conectados:

- El núcleo de Greengrass se apaga.
- Todos los componentes que se ejecutan en el dispositivo principal también se apagan. Si estos componentes escriben en recursos locales, es posible que dejen esos recursos en un estado incorrecto, salvo que se apaguen correctamente. Los componentes pueden utilizar la comunicación entre procesos para indicar al componente del núcleo que aplace la actualización hasta que borren los recursos que utilizan.
- Mientras el componente del núcleo está apagado, el dispositivo principal pierde sus conexiones con Nube de AWS y con los dispositivos locales. El dispositivo principal no enrutará los mensajes de los dispositivos de cliente mientras esté apagado.
- Las funciones de Lambda de larga duración que se ejecutan como componentes perderán la información de estado dinámico y eliminarán todos los trabajos pendientes.

Comportamiento de actualización del núcleo de Greengrass

Cuando implementa un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos proporcionados por AWS se implementen de forma automática en sus dispositivos principales si agrega nuevos dispositivos a un grupo de elementos o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.

Cuando la versión del componente de núcleo de Greengrass cambia, el software AWS IoT Greengrass Core, que incluye el núcleo y todos los demás componentes del dispositivo, se reinicia para aplicar los cambios. Debido a que la actualización del componente núcleo afecta a los dispositivos principales, es posible que desee controlar cuándo se implementará una nueva versión del parche de núcleo en sus dispositivos. Para ello, debe incluir directamente el componente de núcleo de Greengrass en su implementación. Incluir directamente un componente significa incluir una versión específica de ese componente en la configuración de implementación y no depender de las dependencias de los componentes para implementar ese componente en sus dispositivos. Para obtener más información sobre cómo definir dependencias en sus recetas de componentes, consulte Formato de receta.

Consulte la siguiente tabla para comprender el comportamiento de actualización del componente de núcleo de Greengrass en función de sus acciones y configuraciones de implementación.

Acción	Configuración de implement ación	Comportamiento de actualiza ción de núcleo
Agregue nuevos dispositi vos a un grupo de objetos al que apunta una implement ación existente sin revisar la implementación.	La implementación no incluye directamente el núcleo de Greengrass. La implementación incluye directamente al menos un componente proporcio nado por AWS o incluye un componente personali zado que depende de un componente proporcionado	En los dispositivos nuevos, instala la última versión de parche de núcleo que cumple con todos los requisito s de dependencia de los componentes. En los dispositivos existentes, no actualiza la versión instalada del núcleo.

Acción	Configuración de implement ación	Comportamiento de actualiza ción de núcleo
	por AWS o del núcleo de Greengrass.	
Agregue nuevos dispositi vos a un grupo de objetos al que apunta una implement ación existente sin revisar la implementación.	La implementación incluye directamente una versión específica del núcleo de Greengrass.	En los dispositivos nuevos, instala la versión del núcleo especificada. En los dispositivos existentes, no actualiza la versión instalada del núcleo.
Cree una nueva implement ación o revise una implement ación existente.	La implementación no incluye directamente el núcleo de Greengrass. La implementación incluye directamente al menos un componente proporcio nado por AWS o incluye un componente personali zado que depende de un componente proporcionado por AWS o del núcleo de Greengrass.	En todos los dispositivos de destino, instala la última versión del parche del núcleo que cumpla con todos los requisitos de dependencia de los componentes, incluso en los dispositivos nuevos que agregue al grupo de objetos de destino.
Cree una nueva implement ación o revise una implement ación existente.	La implementación incluye directamente una versión específica del núcleo de Greengrass.	En todos los dispositivos de destino, instala la versión de núcleo especificada, incluidos los dispositivos nuevos que agregue al grupo de objetos de destino.

Cómo realizar actualizaciones OTA

Para realizar una actualización OTA, cree una implementación que incluya el componente de núcleo y la versión que desee instalar.

Desinstalación del software AWS IoT Greengrass Core

Puede desinstalar el software AWS IoT Greengrass Core para eliminarlo de un dispositivo que no desee utilizar como dispositivo principal de Greengrass. También puede seguir estos pasos para limpiar una instalación que no funcione.

Cómo desinstalar el software AWS IoT Greengrass Core

1. Si ejecuta el software como un servicio del sistema, debe detener, deshabilitar y eliminar el servicio. Ejecute los comandos apropiados para su sistema operativo.

   Linux

   1. Detenga el servicio .

      ```
      sudo systemctl stop greengrass.service
      ```

   2. Deshabilite el servicio.

      ```
      sudo systemctl disable greengrass.service
      ```

   3. Elimine el servicio.

      ```
      sudo rm /etc/systemd/system/greengrass.service
      ```

   4. Compruebe que el servicio se eliminó.

      ```
      sudo systemctl daemon-reload && sudo systemctl reset-failed
      ```

Windows (Command Prompt)

> **Note**
>
> Debe ejecutar Command Prompt o PowerShell como administrador para ejecutar estos comandos.

1. Detenga el servicio .

```
sc stop "greengrass"
```

2. Deshabilite el servicio.

```
sc config "greengrass" start=disabled
```

3. Elimine el servicio.

```
sc delete "greengrass"
```

4. Reinicie el dispositivo.

Windows (PowerShell)

> **Note**
>
> Debe ejecutar PowerShell como administrador para ejecutar estos comandos.

1. Detenga el servicio .

```
Stop-Service -Name "greengrass"
```

2. Deshabilite el servicio.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. Elimine el servicio.

- Para PowerShell 6.0 o posterior:

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

- Para las versiones de PowerShell anteriores a la 6.0:

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item
 -Force -Verbose
```

4. Reinicie el dispositivo.

2. Elimine la carpeta raíz del dispositivo. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta de acceso a la carpeta raíz.

   Linux

   ```
   sudo rm -rf /greengrass/v2
   ```

   Windows (Command Prompt)

   ```
   rmdir /s /q C:\greengrass\v2
   ```

   Windows (PowerShell)

   ```
   cmd.exe /c "rmdir /s /q C:\greengrass\v2"
   ```

3. Elimine el dispositivo principal del servicio de AWS IoT Greengrass. Este paso elimina la información de estado del dispositivo principal de la Nube de AWS. Asegúrese de completar este paso si planea volver a instalar el software AWS IoT Greengrass Core en un dispositivo principal con el mismo nombre.

   - Para eliminar un dispositivo principal de la consola de AWS IoT Greengrass, haga lo siguiente:

     a. Vaya a la consola de AWS IoT Greengrass.

     b. Elija Dispositivos principales.

     c. Elija el dispositivo principal que desee eliminar.

     d. Elija Eliminar.

     e. En el cuadro de confirmación, elija Eliminar.

- Para eliminar un dispositivo principal con la AWS Command Line Interface, utilice la operación DeleteCoreDevice. Ejecute el siguiente comando y reemplace *MyGreengrassCore* por el nombre del dispositivo principal.

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 tutoriales

AWS IoT Greengrass es un servicio que le permite ejecutar AWS Lambda funciones, modelos de aprendizaje automático y otros códigos en dispositivos periféricos. Así, puede procesar los datos de forma local, lo que reduce los costos de latencia y ancho de banda y, al mismo tiempo, mantiene una comunicación segura con la nube.

Puede completar los siguientes tutoriales para obtener información sobre la AWS IoT Greengrass V2 y sus funciones.

Temas

- Tutoriales de vídeo
- Tutorial: Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes
- Tutorial: interactúe con dispositivos IoT locales a través de MQTT
- Tutorial: Proteja Greengrass Nucleus con Trusted Platform Module (TPM)
- Tutorial: Proteja Nucleus Lite de AWS IoT Greengrass con Trusted Platform Module (TPM)
- Tutorial: Cómo empezar a SageMaker usar AI Edge Manager
- Tutorial: Realice una inferencia de clasificación de imágenes de muestra con Lite TensorFlow
- Tutorial: Realice una inferencia de clasificación de imágenes de muestra en imágenes de una cámara con Lite TensorFlow

Tutoriales de vídeo

Videos

La siguiente lista incluye tutoriales en video para AWS IoT Greengrass V2.

- Tutorial de la versión lite del núcleo de AWS IoT Greengrass: introducción

  10 de abril de 2025: en este tutorial, aprenderá sobre la versión lite del núcleo de Greengrass y cuándo seleccionarla para sus dispositivos de AWS IoT. Aprenderá cómo instalar la versión lite del núcleo e implementar sus componentes en ella.
- Tutorial de AWS IoT Greengrass V2: introducción

16 de diciembre de 2021: introducción a AWS IoT Greengrass V2. En este tutorial, aprenderá a instalar Greengrass y a completar una primera implementación.

- Desarrollo de componentes de AWS IoT Greengrass V2 con una consola de depuración local

  21 de agosto de 2023: aprenda a usar la consola de depuración local de la versión 2 de Greengrass para desarrollar y depurar componentes rápidamente.

- Connecte dispositivos cliente MQTT a AWS IoT Greengrass V2

  2 de agosto de 2023: en este tutorial, aprenderá a configurar un dispositivo principal para que interactúe con los dispositivos locales de AWS IoT que se conectan al dispositivo principal mediante MQTT.

- Tutorial de AWS IoT Greengrass V2: Uso de aplicaciones MQTT

  16 de diciembre de 2021: cree, pruebe e implemente un componente nativo de AWS IoT Greengrass V2 para publicar y suscribirse a MQTT a través de AWS IoT Core.

Tutorial: Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes

Puede completar este tutorial para desarrollar un componente que aplaza las actualizaciones de over-the-air implementación. Al implementar actualizaciones en sus dispositivos, es posible que desee aplazar las actualizaciones en función de ciertas condiciones, como las siguientes:

- El nivel de batería del dispositivo es bajo.
- El dispositivo está ejecutando un proceso o un trabajo que no se puede interrumpir.
- El dispositivo tiene una conexión a Internet limitada o cara.

> **Note**
>
> Un componente es un módulo de software que se ejecuta en los dispositivos AWS IoT Greengrass principales. Los componentes le permiten crear y administrar aplicaciones complejas como bloques de compilación discretos que puede reutilizar de un dispositivo principal de Greengrass a otro.

En este tutorial, aprenderá a hacer lo siguiente:

1. Instale la CLI del kit de desarrollo de Greengrass (CLI del GDK) en su computadora de desarrollo. La CLI del GDK proporciona características que lo ayudan a desarrollar componentes personalizados de Greengrass.
2. Desarrolle un componente Hello World que aplace las actualizaciones de los componentes cuando el nivel de batería del dispositivo principal esté por debajo de un umbral. Este componente se suscribe a las notificaciones de actualización mediante la operación SubscribeToComponentUpdatesIPC. Cuando recibe la notificación, comprueba si el nivel de la batería es inferior a un umbral personalizable. Si el nivel de la batería está por debajo del umbral, aplaza la actualización durante 30 segundos mediante la DeferComponentUpdateoperación IPC. Este componente se desarrolla en la computadora de desarrollo mediante la CLI del GDK.

   > **Note**
   >
   > Este componente lee el nivel de la batería a partir de un archivo que usted crea en el dispositivo principal para imitar una batería real, por lo que puede completar este tutorial en un dispositivo principal sin batería.

3. Publique ese componente en el AWS IoT Greengrass servicio.
4. Despliegue ese componente desde Nube de AWS el dispositivo principal de Greengrass para probarlo. A continuación, se modifica el nivel de batería virtual del dispositivo principal y se crean implementaciones adicionales para comprobar cómo el dispositivo principal aplaza las actualizaciones cuando el nivel de batería es bajo.

Calcule dedicar unos 20-30 minutos a este tutorial.

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Un Cuenta de AWS. Si no dispone de una, consulte Configura un Cuenta de AWS.
- Un usuario de AWS Identity and Access Management (IAM) con permisos de administrador.
- Un dispositivo principal de Greengrass con conexión a Internet. Para obtener más información acerca de cómo configurar un dispositivo principal, consulte Configuración de los dispositivos AWS IoT Greengrass principales.

- Se necesita tener la versión 3.6 o posterior de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH. En Windows, también debe tener instalado el lanzador de Python para Windows para todos los usuarios.

  > **Important**
  >
  > En Windows, Python no se instala para todos los usuarios de forma predeterminada. Al instalar Python, debe personalizar la instalación para configurarla para que el software AWS IoT Greengrass Core ejecute scripts de Python. Por ejemplo, si usa el instalador gráfico de Python, haga lo siguiente:
  >
  > 1. Elija Instalar el lanzador para todos los usuarios (recomendado).
  > 2. Elija Customize installation.
  > 3. Elija Next.
  > 4. Seleccione Install for all users.
  > 5. Seleccione Add Python to environment variables.
  > 6. Elija Instalar.
  >
  > Para obtener más información, consulte Uso de Python en Windows en la documentación de Python 3.

- Una computadora de desarrollo similar a Windows, macOS o Unix con conexión a Internet.
  - Versión 3.6 o posterior de Python instalada en su computadora de desarrollo.
  - Git instalado en su computadora de desarrollo.
  - AWS Command Line Interface (AWS CLI) instalado y configurado con credenciales en su computadora de desarrollo. Para obtener más información, consulte Instalar, actualizar y desinstalar la AWS CLI y Configurar la AWS CLI en la Guía del usuario de AWS Command Line Interface.

    > **Note**
    >
    > Si usa una Raspberry Pi u otro dispositivo ARM de 32 bits, instale la versión 1 de AWS CLI. AWS CLI La versión 2 no está disponible para dispositivos ARM de 32 bits. Para obtener más información, consulte Instalar, actualizar y desinstalar la versión 1 de AWS CLI.

Paso 1: Instalar la CLI del kit de desarrollo de Greengrass

La CLI del kit de desarrollo de Greengrass (CLI del GDK) proporciona características que lo ayudan a desarrollar componentes personalizados de Greengrass. Puede usar la CLI del GDK para crear, compilar y publicar componentes personalizados.

Si no ha instalado la CLI del GDK en su equipo de desarrollo, complete los siguientes pasos para instalarla.

Cómo instalar la última versión de la CLI del GDK

1. En su computadora de desarrollo, ejecute el siguiente comando para instalar la versión más reciente de la CLI de GDK desde su repositorio de GitHub.

   ```
   python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
   ```

2. Ejecute los siguientes comandos para verificar que la CLI del GDK se instaló correctamente.

   ```
   gdk --help
   ```

   Si no se encuentra el comando `gdk`, agregue su carpeta a PATH.

   - En dispositivos Linux, agregue `/home/`*`MyUser`*`/.local/bin` a PATH y sustituya *`MyUser`* por el nombre de su usuario.
   - En dispositivos Windows, agregue *`PythonPath`*`\\Scripts` a PATH y sustituya *`PythonPath`* por la ruta a la carpeta Python de su dispositivo.

Paso 2: Desarrollar un componente que aplace las actualizaciones

En esta sección, desarrollará un componente Hello World en Python que aplaza las actualizaciones de los componentes cuando el nivel de batería del dispositivo principal esté por debajo del umbral que usted configuró al implementar el componente. En este componente, se utiliza la interfaz de comunicación entre procesos (IPC) de la SDK para dispositivos con AWS IoT versión 2 para Python. La operación SubscribeToComponentUpdatesIPC se utiliza para recibir notificaciones cuando el dispositivo principal recibe una implementación. A continuación, se utiliza la operación DeferComponentUpdateIPC para aplazar o confirmar la actualización en función del nivel de batería del dispositivo.

Desarrollo de un componente Hello World que aplace las actualizaciones

1. En su computadora de desarrollo, cree una carpeta para el código de origen del componente.

   ```
   mkdir com.example.BatteryAwareHelloWorld
   cd com.example.BatteryAwareHelloWorld
   ```

2. Utilice un editor de texto para crear un archivo llamado `gdk-config.json`. La CLI de GDK lee el archivo de configuración de la CLI de GDK, denominado `gdk-config.json`, para crear y publicar componentes. Este archivo de configuración existe en la raíz de la carpeta del componente.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano gdk-config.json
   ```

   Copie el siguiente JSON en el archivo.

   - *Amazon*Sustitúyalo por tu nombre.
   - *us-west-2*Reemplácelo por el Región de AWS lugar donde funciona su dispositivo principal. La CLI del GDK publica el componente en esta Región de AWS.
   - *greengrass-component-artifacts*Sustitúyalo por el prefijo de bucket S3 que desee utilizar. Cuando utiliza la CLI de GDK para publicar el componente, la CLI de GDK carga los artefactos del componente en el bucket de S3 cuyo nombre se forma a partir de este valor Región de AWS, el y su Cuenta de AWS ID con el siguiente formato:. *bucketPrefix*-*region*-*accountId*

     Por ejemplo, si especificas **greengrass-component-artifacts** y**us-west-2**, y tu Cuenta de AWS ID es**123456789012**, la CLI de GDK usa el bucket de S3 denominado`greengrass-component-artifacts-us-west-2-123456789012`.

   ```
   {
     "component": {
       "com.example.BatteryAwareHelloWorld": {
         "author": "Amazon",
         "version": "NEXT_PATCH",
         "build": {
   ```

```
        "build_system" : "zip"
      },
      "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

El archivo de configuración especifica lo siguiente:

- La versión que se utilizará cuando la CLI de GDK publique el componente Greengrass en el servicio en AWS IoT Greengrass la nube. `NEXT_PATCH`especifica que se debe elegir la siguiente versión del parche después de la última versión disponible en el servicio en la AWS IoT Greengrass nube. Si el componente aún no tiene una versión en el servicio en la AWS IoT Greengrass nube, la CLI de GDK la utiliza`1.0.0`.
- El sistema de compilación del componente. Cuando se utiliza el sistema de compilación `zip`, la CLI del GDK empaqueta el código de origen del componente en un archivo ZIP que se convierte en el único artefacto del componente.
- Allí Región de AWS donde la CLI de GDK publica el componente Greengrass.
- El prefijo del bucket de S3 donde la CLI del GDK carga los artefactos del componente.

3. Utilice un editor de texto para crear el código de origen del componente en un archivo denominado `main.py`.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano main.py
   ```

   Copie el siguiente código de Python en el archivo.

   ```
   import json
   import os
   import sys
   import time
   import traceback
   ```

```
from pathlib import Path

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15  # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000  # Milliseconds


class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
 Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                          event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                          event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
            # SubscribeToComponentUpdates returns a tuple with the response and the
 operation.
            _, self.subscription_operation =
 self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
        if self.subscription_operation is not None:
            self.subscription_operation.close()
```

```
            self.subscription_operation = None

    def defer_update(self, deployment_id):
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id,
 recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

    def acknowledge_update(self, deployment_id):
        # Specify recheck_after_ms=0 to acknowledge a component update.
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id, recheck_after_ms=0)

    def is_battery_below_threshold(self):
        return self.get_battery_level() < self.battery_threshold

    def get_battery_level(self):
        # Read the battery level from the virtual battery level file.
        with self.battery_file_path.open('r') as f:
            data = json.load(f)
            return float(data['battery_level'])

    def print_message(self):
        message = 'Hello, World!'
        if self.is_battery_below_threshold():
            message += ' Battery level (%d) is below threshold (%d), so the
 component will defer updates' % (
                self.get_battery_level(), self.battery_threshold)
        else:
            message += ' Battery level (%d) is above threshold (%d), so the
 component will acknowledge updates' % (
                self.get_battery_level(), self.battery_threshold)
        print(message)


def main():
    # Read the battery threshold and virtual battery file path from command-line
 args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))

    try:
```

```
        # Create an IPC client and a Hello World printer that defers component
 updates.
        ipc_client = GreengrassCoreIPCClientV2()
        hello_world_printer = BatteryAwareHelloWorldPrinter(
            ipc_client, battery_file_path, battery_threshold)
        hello_world_printer.subscribe_to_component_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                hello_world_printer.print_message()
                time.sleep(HELLO_WORLD_PRINT_INTERVAL)
        except InterruptedError:
            print('Subscription interrupted')
        hello_world_printer.close_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

Esta aplicación de Python hace lo siguiente:

- Lee el nivel de batería del dispositivo principal a partir de un archivo de nivel de batería virtual que creará en el dispositivo principal más adelante. Este archivo de nivel de batería virtual imita una batería real, por lo que puede completar este tutorial en los dispositivos principales que no tienen batería.
- Lee los argumentos de la línea de comandos para el umbral de batería y la ruta al archivo de nivel de batería virtual. La receta del componente establece estos argumentos de la línea de comandos en función de los parámetros de configuración, por lo que puede personalizar estos valores al implementar el componente.
- Utiliza el cliente IPC V2 en la SDK para dispositivos con AWS IoT versión 2 para Python para comunicarse con el software AWS IoT Greengrass Core. En comparación con el cliente IPC original, la versión 2 del cliente IPC reduce la cantidad de código que hay que escribir para usar el IPC en componentes personalizados.
- Se suscribe para actualizar las notificaciones mediante la operación SubscribeToComponentUpdatesIPC. El software AWS IoT Greengrass principal envía notificaciones antes y después de cada implementación. El componente llama a la siguiente

función cada vez que recibe una notificación. Si la notificación es para una próxima implementación, el componente comprueba si el nivel de la batería es inferior a un umbral. Si el nivel de la batería está por debajo del umbral, el componente aplaza la actualización durante 30 segundos mediante la operación DeferComponentUpdateIPC. De lo contrario, si el nivel de la batería no está por debajo del umbral, el componente confirma la actualización para que esta pueda continuar.

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

> **Note**
>
> El software AWS IoT Greengrass principal no envía notificaciones de actualización para las implementaciones locales, por lo que debe implementar este componente mediante el servicio AWS IoT Greengrass en la nube para probarlo.

4. Utilice un editor de texto para crear la receta del componente en un archivo denominado `recipe.json` o `recipe.yaml`. La receta del componente define los metadatos del componente, parámetros de configuración predeterminados y scripts de ciclo de vida específicos de la plataforma.

   JSON

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano recipe.json
```

Copie el siguiente JSON en el archivo.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "COMPONENT_NAME",
  "ComponentVersion": "COMPONENT_VERSION",
  "ComponentDescription": "This Hello World component defers updates when the
 battery level is below a threshold.",
  "ComponentPublisher": "COMPONENT_AUTHOR",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "BatteryThreshold": 50,
      "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
      "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk --upgrade",
        "Run": "python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/LinuxBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
```

```
        "install": "py -3 -m pip install --user awsiotsdk --upgrade",
        "Run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/WindowsBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

YAML

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano recipe.yaml
```

Copie el siguiente YAML en el archivo.

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
 battery level is below a threshold."
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade
```

```
      Run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk --upgrade
      Run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/WindowsBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
```

Esta receta especifica lo siguiente:

- Parámetros de configuración predeterminados para el umbral de batería, la ruta del archivo de batería virtual en los dispositivos principales de Linux y la ruta del archivo de batería virtual en los dispositivos principales de Windows.
- Un ciclo de vida `install` que instala la última versión de la versión 2 de SDK para dispositivos con AWS IoT para Python.
- Un ciclo de vida `run` que ejecuta la aplicación Python en `main.py`.
- Marcadores de posición, como `COMPONENT_NAME` y `COMPONENT_VERSION`, donde la CLI del GDK reemplaza la información al crear la receta del componente.

Para obtener más información sobre las recetas de componentes, consulte AWS IoT Greengrass referencia de recetas de componentes.

Paso 3: Publicar el componente en el AWS IoT Greengrass servicio

En esta sección, publica el componente Hello World en el servicio AWS IoT Greengrass en la nube. Una vez que un componente esté disponible en el servicio en la AWS IoT Greengrass nube, puede implementarlo en los dispositivos principales. Utilice la CLI de GDK para publicar el componente

desde su ordenador de desarrollo en el servicio AWS IoT Greengrass en la nube. La CLI del GDK carga la receta y los artefactos del componente por usted.

Para publicar el componente Hello World en el servicio AWS IoT Greengrass

1. Ejecute el siguiente comando para compilar el componente mediante la CLI del GDK. El comando component build crea una receta y artefactos basados en el archivo de configuración CLI de GDK. En este proceso, la CLI de GDK crea un archivo ZIP que contiene el código de origen del componente.

   ```
   gdk component build
   ```

   Debería ver mensajes similares al del siguiente ejemplo.

   ```
   [2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:16] INFO - Building the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
   [2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
    system. If this is incorrect, please exit and specify custom build command in the
    'gdk-config.json'.
   [2022-04-28 11:20:16] INFO - Zipping source code files of the component.
   [2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
    component artifacts build folder.
   [2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
   [2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
   \greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
   ```

2. Ejecute el siguiente comando para publicar el componente en el servicio AWS IoT Greengrass en la nube. El comando component publish carga el artefacto del archivo ZIP del componente en un bucket de S3. A continuación, actualiza el URI de S3 del archivo ZIP en la receta del componente y carga la receta en el servicio de AWS IoT Greengrass . En este proceso, la CLI de GDK comprueba qué versión del componente Hello World ya está disponible en el servicio en la AWS IoT Greengrass nube para poder elegir la siguiente versión del parche después de esa versión. Si el componente aún no existe, la CLI de GDK usa la versión `1.0.0`.

   ```
   gdk component publish
   ```

Debería ver mensajes similares al del siguiente ejemplo. El resultado indica la versión del componente que creó la CLI de GDK.

```
[2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2022-04-28 11:20:30] INFO - No private version of the component
 'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
 next version to create.
[2022-04-28 11:20:30] INFO - Publishing the component
 'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
[2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
 bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
 first time using this bucket, add the 's3:GetObject' permission to each core
 device's token exchange role to allow it to download the component artifacts. For
 more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
device-service-role.html.
[2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
[2022-04-28 11:20:30] INFO - Updating the component recipe
 com.example.BatteryAwareHelloWorld-1.0.0.
[2022-04-28 11:20:31] INFO - Creating a new greengrass component
 com.example.BatteryAwareHelloWorld-1.0.0
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
 the account.'com.example.BatteryAwareHelloWorld'.
```

3. Copie el nombre del bucket de S3 de la salida. El nombre del bucket se utiliza más adelante para permitir que el dispositivo principal descargue los artefactos componentes de este bucket.
4. (Opcional) Vea el componente en la AWS IoT Greengrass consola para comprobar que se ha cargado correctamente. Haga lo siguiente:

   a. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.

   b. En la página Componentes, elija la pestaña Mis componentes y, a continuación, elija com.example.BatteryAwareHelloWorld.

      En esta página, puede ver la receta del componente y otra información sobre el componente.

5. Permita que el dispositivo principal acceda a los artefactos de los componentes del bucket de S3.

   Cada dispositivo principal tiene una función de IAM del dispositivo principal que le permite interactuar con los registros AWS IoT y enviarlos a la AWS nube. Este rol de dispositivo no permite el acceso a los bucket de S3 de forma predeterminada, por lo que debe crear y adjuntar una política que permita que el dispositivo principal recupere artefactos de componentes del bucket de S3.

   Puede omitir este paso si el rol de su dispositivo ya permite acceder al bucket de S3. De lo contrario, cree una política de IAM que permita el acceso y adjúntela al rol, de la siguiente manera:

   a. En el panel de navegación de la consola de IAM, elija Políticas, seguido de Crear política.

   b. En la pestaña JSON, reemplace el contenido del marcador de posición por la política siguiente. *greengrass-component-artifacts-us-west-2-123456789012*Sustitúyalo por el nombre del depósito de S3 en el que la CLI de GDK cargó los artefactos del componente.

      Por ejemplo, si especificó **greengrass-component-artifacts** y **us-west-2** en el archivo de configuración de la CLI de GDK y su ID de Cuenta de AWS es **123456789012**, la CLI de GDK usa el bucket de S3 llamado `greengrass-component-artifacts-us-west-2-123456789012`.

      JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-west-2-123456789012/*"
    }
  ]
}
```

c. Elija Siguiente.

d. En la sección Detalles de política, en Nombre, introduzca **`MyGreengrassV2ComponentArtifactPolicy`**.

e. Elija Crear política.

f. En el menú de navegación de la consola de IAM, seleccione Rol y, a continuación, elija el nombre del rol para el dispositivo principal. Especificó este nombre de función al instalar el software AWS IoT Greengrass Core. Si no especifica un nombre, el nombre predeterminado es `GreengrassV2TokenExchangeRole`.

g. En la pestaña Permisos, elija Agregar permisos y, a continuación, Asociar políticas.

h. En la sección Otras políticas de permisos, seleccione la casilla de verificación junto a la política `MyGreengrassV2ComponentArtifactPolicy` que creó, a continuación, elija Agregar permisos.

Paso 4: Implementación y prueba del componente en un dispositivo principal

En esta sección, implementará el componente en el dispositivo principal para probar su funcionalidad. En el dispositivo principal, se crea el archivo de nivel de batería virtual para imitar una batería real. A continuación, debe crear implementaciones adicionales y observar los archivos de registro de los componentes del dispositivo principal para comprobar si el componente se aplaza y confirma las actualizaciones.

Implementación y prueba del componente Hello World que aplaza las actualizaciones

1. Use un editor de texto para crear un archivo sobre la batería virtual. Este archivo imita una batería real.

   - En los dispositivos principales de Linux, cree un archivo con el nombre `/home/ggc_user/virtual_battery.json`. Ejecute el editor de texto con permisos `sudo`.
   - En los dispositivos principales de Windows, cree un archivo con el nombre `C:\Users\ggc_user\virtual_battery.json`. Ejecute el editor de texto como administrador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
sudo nano /home/ggc_user/virtual_battery.json
```

Copie el siguiente JSON en el archivo.

```
{
  "battery_level": 50
}
```

2. Implemente el componente Hello World en el dispositivo principal. Haga lo siguiente:

   a. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.

   b. En la página Componentes, elija la pestaña Mis componentes y, a continuación, elija com.example.BatteryAwareHelloWorld.

   c. En la página com.example.BatteryAwareHelloWorld, elija Implementar.

   d. En Agregar a la implementación, elija una implementación existente para revisarla o cree una nueva y, a continuación, elija Siguiente.

   e. Si opta por crear una nueva implementación, elija el dispositivo principal o el grupo de objetos de destino para la implementación. En la página Especificar el destino, en Destino de la implementación, elija un dispositivo principal o un grupo de objetos y, a continuación, elija Siguiente.

   f. En la página Seleccionar componentes, compruebe que el componente com.example.BatteryAwareHelloWorld esté seleccionado y elija Siguiente.

   g. En la página Configurar componentes, seleccione com.example.BatteryAwareHelloWorld y haga lo siguiente:

      i. Seleccione Configurar componente.

      ii. En el cuadro Configurar com.example.BatteryAwareHelloWorld, en Actualización de configuración, en Configuración para combinar, ingrese la siguiente actualización de configuración.

      ```
      {
        "BatteryThreshold": 70
      }
      ```

      iii. Elija Confirmar para cerrar el cuadro y, a continuación, elija Siguiente.

h. En la página Confirmar la configuración avanzada, en la sección Políticas de implementación, en Política de actualización de componentes, confirme que está seleccionada la opción Notificar componentes. La opción Notificar componentes está seleccionada de forma predeterminada al crear una nueva implementación.

i. En la página Revisar, elija Implementar.

La implementación puede tardar hasta un minuto para completarse.

3. El software AWS IoT Greengrass Core guarda la salida estándar de los procesos de los componentes en los archivos de registro de la `logs` carpeta. Ejecute el siguiente comando para verificar que el componente Hello World ejecuta e imprime los mensajes de estado.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
```

Debería ver mensajes similares al del siguiente ejemplo.

```
Hello, World! Battery level (50) is below threshold (70), so the component will
 defer updates.
```

> **Note**
>
> Si el archivo no existe, es posible que la implementación aún no esté completa. Si el archivo no existe en 30 segundos, es probable que la implementación haya fallado. Esto puede ocurrir si el dispositivo principal no tiene permiso para descargar los artefactos del componente desde el bucket de S3, por ejemplo. Ejecute el siguiente comando para ver el archivo de registro del software AWS IoT Greengrass principal. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.log
```

El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

4. Cree una nueva implementación en el dispositivo principal para comprobar que el componente aplaza la actualización. Haga lo siguiente:

   a. En el menú de navegación de la consola de AWS IoT Greengrass, elija Implementaciones.

   b. Elija la implementación que creó o revisó anteriormente.

   c. En la página de implementación, elija Revisar.

   d. En el cuadro Revisar implementación, elija Revisar implementación.

   e. Elija Siguiente en cada paso y, a continuación, elija Implementar.

5. Ejecute el siguiente comando para ver de nuevo los registros del componente y comprobar que aplaza la actualización.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
```

Debería ver mensajes similares al del siguiente ejemplo. El componente aplaza la actualización durante 30 segundos, por lo que imprime este mensaje repetidamente.

```
Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
```

6. Use un editor de texto para editar el archivo de la batería virtual y cambie el nivel de batería a un valor superior al umbral, de modo que la implementación pueda continuar.

   - En los dispositivos principales de Linux, edite el archivo denominado `/home/ggc_user/virtual_battery.json`. Ejecute el editor de texto con permisos `sudo`.
   - En los dispositivos principales de Windows, edite el archivo denominado `C:\Users\ggc_user\virtual_battery.json`. Ejecute el editor de texto como administrador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   Cambie el nivel de la batería a `80`.

   ```
   {
     "battery_level": 80
   }
   ```

7. Ejecute el siguiente comando para ver de nuevo los registros del componente y comprobar que reconoce la actualización.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
```

Debería ver mensajes similares a los de los siguientes ejemplos.

```
Hello, World! Battery level (80) is above threshold (70), so the component will
 acknowledge updates.
Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
```

Completó este tutorial. El componente Hello World aplaza o confirma las actualizaciones en función del nivel de batería del dispositivo principal. Para obtener más información sobre lo que se incluye en este tutorial, consulte lo siguiente:

- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core
- Interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass

Tutorial: interactúe con dispositivos IoT locales a través de MQTT

Puede completar este tutorial para configurar un dispositivo principal para que interactúe con los dispositivos IoT locales, denominados dispositivos de cliente, que se conectan al dispositivo principal a través de MQTT. En este tutorial, configurará AWS IoT las cosas para usar la detección en la nube para conectarse al dispositivo principal como dispositivos cliente. Al configurar la detección en la nube, un dispositivo cliente puede enviar una solicitud al servicio AWS IoT Greengrass en la nube para descubrir los dispositivos principales. La respuesta de AWS IoT Greengrass incluye la información de conectividad y los certificados de los dispositivos principales que usted configura para

que los detecte el dispositivo de cliente. Luego, el dispositivo de cliente puede usar esta información para conectarse a un dispositivo principal disponible donde pueda comunicarse a través de MQTT.

En este tutorial, aprenderá a hacer lo siguiente:

1. Revise y actualice los permisos del dispositivo principal, si es necesario.
2. Asocié los dispositivos de cliente al dispositivo principal para que puedan detectar el dispositivo principal mediante la detección en la nube.
3. Implemente los componentes de Greengrass en el dispositivo principal para permitir la compatibilidad con el dispositivo de cliente.
4. Conecte los dispositivos de cliente al dispositivo principal y pruebe la comunicación con el servicio en la nube de AWS IoT Core .
5. Desarrolle un componente de Greengrass personalizado que se comunique con los dispositivos de cliente.
6. Desarrolle un componente personalizado que interactúe con las sombras de dispositivos de cliente de AWS IoT.

Este tutorial usa un dispositivo principal único y un dispositivo de cliente único. También puede seguir el tutorial para conectar y probar varios dispositivos de cliente.

Calcule dedicar unos 30 a 60 minutos a este tutorial.

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Un Cuenta de AWS. Si no dispone de una, consulte Configura un Cuenta de AWS.
- Un usuario AWS Identity and Access Management (IAM) con permisos de administrador.
- Un dispositivo principal de Greengrass. Para obtener más información acerca de cómo configurar un dispositivo principal, consulte Configuración de los dispositivos AWS IoT Greengrass principales.
  - El dispositivo principal debe ejecutar la versión 2.6.0 o posterior del núcleo de Greengrass. Esta versión incluye la compatibilidad con los caracteres comodín en la publish/subscribe comunicación local y la compatibilidad con los dispositivos cliente ocultos.

> **Note**
>
> La compatibilidad con dispositivos de cliente requiere el núcleo de Greengrass versión 2.2.0 o posterior. Sin embargo, en este tutorial se analizan las funciones más recientes, como la compatibilidad con los caracteres comodín de MQTT en el entorno local publish/subscribe y la compatibilidad con las sombras de los dispositivos cliente. Estas características requieren el núcleo de Greengrass versión 2.6.0 o posterior.

  - El dispositivo principal debe estar en la misma red que los dispositivos de cliente para conectarse.
  - (Opcional) Para completar los módulos en los que se desarrollan componentes personalizados de Greengrass, el dispositivo principal debe ejecutar la CLI de Greengrass. Para obtener más información, consulte Instalación de la CLI de Greengrass.
- En este tutorial, AWS IoT hay algo que conectar como dispositivo cliente. Para obtener más información, consulte Crear AWS IoT recursos en la Guía para AWS IoT Core desarrolladores.
  - La AWS IoT política del dispositivo cliente debe permitir el `greengrass:Discover` permiso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos cliente.
  - El dispositivo de cliente debe estar en la misma red que el dispositivo principal.
  - El dispositivo de cliente debe ejecutar Python 3.
  - El dispositivo de cliente debe ejecutar Git.

Paso 1: Revise y actualice la AWS IoT política principal de dispositivos

Para ser compatible con los dispositivos cliente, la AWS IoT política de un dispositivo principal debe permitir los siguientes permisos:

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (Opcional) Este permiso es necesario para utilizar el componente de detección de IP, que envía la información de conectividad de red del dispositivo principal al servicio AWS IoT Greengrass en la nube.

Para obtener más información sobre estos permisos y AWS IoT políticas para los dispositivos principales, consulte AWS IoT políticas para las operaciones del plano de datos yAWS IoT Política mínima de compatibilidad con los dispositivos cliente.

En esta sección, revisas las AWS IoT políticas de tu dispositivo principal y añades los permisos necesarios que falten. Si utilizaste el instalador del software AWS IoT Greengrass principal para aprovisionar recursos, tu dispositivo principal tiene una AWS IoT política que permite el acceso a todas AWS IoT Greengrass las acciones (`greengrass:*`). En este caso, solo debe actualizar la AWS IoT política si planea configurar el componente de administrador de sombras con el que sincronizar las sombras de los dispositivos AWS IoT Core. De lo contrario, puede omitir esta sección.

Para revisar y actualizar la AWS IoT política de un dispositivo principal

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Dispositivos principales.
2. En la página Dispositivos principales, elija el dispositivo principal que desea actualizar.
3. En la página de detalles del dispositivo principal, elija el enlace al Objeto del dispositivo principal. Este enlace abre la página de detalles del objeto en la consola de AWS IoT .
4. En la página de detalles del objeto, elija Certificados.
5. En la pestaña Certificados, elija el certificado activo del objeto.
6. En la página de detalles del certificado, elija Políticas.
7. En la pestaña Políticas, selecciona la AWS IoT política que deseas revisar y actualizar. Puede agregar los permisos necesarios a cualquier política que esté asociada al certificado activo del dispositivo principal.

   > **Note**
   >
   > Si ha utilizado el instalador de software AWS IoT Greengrass principal para aprovisionar recursos, tiene dos AWS IoT políticas. Le recomendamos que elija la política denominada GreengrassV2IoTThingPolicy, si existe. Los dispositivos principales que cree con el instalador rápido usan este nombre de política de forma predeterminada. Si agrega permisos a esta política, también los otorga a otros dispositivos principales que usan esta política.

8. En la descripción general de la política, elija Editar la versión activa.
9. Revise la política para ver los permisos necesarios y agregue los permisos necesarios que falten.

10. Para establecer una nueva versión de la política como la versión activa, en Estado de la versión de la política, seleccione Establecer la versión editada como la versión activa de esta política.
11. Seleccione Guardar como versión nueva.

Paso 2: Habilitar la compatibilidad del dispositivo de cliente

Para que un dispositivo de cliente use la detección en la nube para conectarse a un dispositivo principal, debe asociar los dispositivos. Cuando asocia un dispositivo de cliente a un dispositivo principal, permite que ese dispositivo de cliente recupere las direcciones IP y los certificados del dispositivo principal para usarlos en la conexión.

Para permitir que los dispositivos cliente se conecten de forma segura a un dispositivo principal y se comuniquen con los componentes de Greengrass AWS IoT Core, debe implementar los siguientes componentes de Greengrass en el dispositivo principal:

- Autenticación del dispositivo de cliente (`aws.greengrass.clientdevices.Auth`)

  Implemente el componente de autenticación del dispositivo de cliente para autenticar los dispositivos de cliente y autorizar las acciones de los dispositivos de cliente. Este componente permite que sus AWS IoT cosas se conecten a un dispositivo principal.

  Este componente requiere alguna configuración para poder usarlo. Debe especificar los grupos de dispositivos de cliente y las operaciones que cada grupo está autorizado a realizar, como conectarse y comunicarse a través de MQTT. Para obtener más información, consulte configuración del componente de autenticación del dispositivo de cliente.
- Agente MQTT 3.1.1 (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Implemente el componente agente MQTT de Moquette para ejecutar un agente MQTT ligero. El agente MQTT de Moquette es compatible con MQTT 3.1.1 e incluye compatibilidad local para QoS 0, QoS 1, QoS 2, mensajes retenidos, mensajes de última voluntad y suscripciones persistentes.

  No es necesario configurar este componente para usarlo. Sin embargo, puede configurar el puerto donde este componente hace funcionar el agente MQTT. Usa el puerto 8883 de manera predeterminada.
- Puente MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`)

(Opcional) Implemente el componente puente MQTT para retransmitir mensajes entre los dispositivos cliente (MQTT local), la publicación/suscripción local y el MQTT. AWS IoT Core Configure este componente para sincronizar los dispositivos cliente AWS IoT Core e interactuar con los dispositivos cliente de los componentes de Greengrass.

Para poder usar este componente, es necesario configurarlo. Debe especificar las asignaciones de temas en las que este componente retransmite los mensajes. Para obtener más información, consulte la configuración del componente puente de MQTT.

- Detector de IP (`aws.greengrass.clientdevices.IPDetector`)

  (Opcional) Implemente el componente detector de IP para informar automáticamente al servicio en la nube de los puntos finales del agente MQTT del AWS IoT Greengrass dispositivo principal. No puede usar este componente si tiene una configuración de red compleja, como una en la que un enrutador reenvía el puerto agente MQTT al dispositivo principal.

  No es necesario configurar este componente para usarlo.

En esta sección, utilizará la AWS IoT Greengrass consola para asociar los dispositivos cliente e implementar los componentes del dispositivo cliente en un dispositivo principal.

Habilitación de la compatibilidad con el dispositivo de cliente

1. Vaya a la consola de AWS IoT Greengrass.
2. En el panel de navegación de la izquierda, elija Dispositivos principales.
3. En la página de Dispositivos principales, elija el dispositivo principal en el que desee habilitar la compatibilidad con el dispositivo de cliente.
4. En la página de detalles del dispositivo principal, elija la pestaña Dispositivos de cliente.
5. En la pestaña Dispositivos de cliente, elija Configurar la detección en la nube.

   Se abre la página Configurar la detección de dispositivos principales. En esta página, puede asociar los dispositivos de cliente a un dispositivo principal e implementar los componentes del dispositivo de cliente. Esta página selecciona el dispositivo principal por usted en el Paso 1: Seleccionar los dispositivos principales de destino.

> **Note**
>
> Puede usar esta página para configurar la detección del dispositivo principal para un grupo de objetos. Si elige esta opción, puede implementar los componentes del dispositivo de cliente en todos los dispositivos principales de un grupo de objetos. Sin embargo, si elige esta opción, deberá asociar manualmente los dispositivos de cliente a cada dispositivo principal más adelante, después de crear la implementación. En este tutorial, configurará un dispositivo principal único.

6. En el paso 2: Asociar los dispositivos cliente, asocie AWS IoT lo del dispositivo cliente al dispositivo principal. Esto permite que el dispositivo de cliente use la detección en la nube para recuperar la información de conectividad y los certificados del dispositivo principal. Haga lo siguiente:

   a. Elija Asociar dispositivos de cliente.

   b. En el modal Asociar dispositivos cliente al dispositivo principal, introduzca el nombre del elemento AWS IoT que se va a asociar.

   c. Elija Add (Añadir).

   d. Elija Associate (Asociar).

7. En el Paso 3: Configurar e implementar los componentes de Greengrass, implemente componentes para habilitar la compatibilidad con el dispositivo de cliente. Si el dispositivo principal de destino tiene una implementación anterior, esta página revisa esa implementación. De lo contrario, esta página crea una nueva implementación para el dispositivo principal. Haga lo siguiente para configurar e implementar los componentes del dispositivo de cliente:

   a. El dispositivo principal debe ejecutar el núcleo de Greengrass versión 2.6.0 o posterior para completar este tutorial. Si el dispositivo principal ejecuta una versión anterior, haga lo siguiente:

      i. Elija la casilla para implementar el componente aws.greengrass.Nucleus.

      ii. Para el componente aws.greengrass.Nucleus, elija Editar configuración.

      iii. Para la versión del componente, elija la versión 2.6.0 o posterior.

      iv. Elija Confirmar.

> **Note**
>
> Si actualiza el núcleo de Greengrass desde una versión secundaria anterior y el dispositivo principal ejecuta componentes AWS proporcionados que dependen del núcleo, también debe actualizar los componentes AWS proporcionados a versiones más recientes. Puede configurar la versión de estos componentes al revisar la implementación más adelante en este tutorial. Para obtener más información, consulte Actualización del software AWS IoT Greengrass Core (OTA).

b. Para el componente aws.greengrass.clientdevices.Auth, elija Editar configuración.

c. En el cuadro de Editar configuración del componente de autenticación del dispositivo de cliente, configure una política de autorización que permita a los dispositivos de cliente publicar y suscribirse al agente MQTT en el dispositivo principal. Haga lo siguiente:

   i. En Configuración, en el bloque de códigos Configuración para combinar, introduzca la siguiente configuración, que contiene una política de autorización de dispositivos de cliente. Cada política de autorización de grupos de dispositivos especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

      - Esta política permite que los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice` se conecten y comuniquen en todos los temas MQTT. *`MyClientDevice*`*Sustitúyalo por el nombre del dispositivo AWS IoT que se va a conectar como dispositivo cliente. También puede especificar un nombre con un comodín `*` que coincida con el nombre del dispositivo de cliente. El comodín `*` debe estar al final del nombre.

        Si tiene que conectar un segundo dispositivo cliente, sustitúyalo *`MyOtherClientDevice*`* por el nombre de ese dispositivo cliente o por un patrón comodín que coincida con el nombre de ese dispositivo cliente. De lo contrario, puede eliminar o conservar esta sección de la regla de selección que permite que los dispositivos de cliente con nombres `MyOtherClientDevice*` que coincidan se conecten y se comuniquen.

      - Esta política usa un operador `OR` para permitir también que los dispositivos de cliente cuyos nombres comiencen por `MyOtherClientDevice` se conecten y se comuniquen en todos los temas de MQTT. Puede eliminar esta cláusula de la regla

de selección o modificarla para que coincida con los dispositivos de cliente a los que se va a conectar.

- Esta política permite que los dispositivos de cliente publiquen y se suscriban en todos los temas MQTT. Para seguir las prácticas recomendadas de seguridad, limite las operaciones `mqtt:publish` y `mqtt:subscribe` al conjunto mínimo de temas que usan los dispositivos de cliente para comunicarse.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice* OR
 thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish to all
 topics.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to all
 topics.",
```

```
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Para obtener más información, consulte Configuración del componente de autenticación del dispositivo de cliente.

ii. Elija Confirmar.

d. Para el componente aws.greengrass.clientdevices.mqtt.Bridge, elija Editar configuración.

e. En el cuadro Editar configuración del componente puente de MQTT, configure una asignación de temas que retransmita los mensajes MQTT de los dispositivos de cliente a AWS IoT Core. Haga lo siguiente:

i. En Configuración, en bloque de códigos Configuración para combinar, introduzca la siguiente configuración. Esta configuración especifica la retransmisión de los mensajes MQTT del filtro de temas `clients/+/hello/world` desde los dispositivos de cliente al servicio en la nube AWS IoT Core . Por ejemplo, este filtro de temas coincide con el tema `clients/MyClientDevice1/hello/world`.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Para obtener más información, consulte la configuración del componente puente de MQTT.

ii. Elija Confirmar.

8. Elija Revisar e implementar para revisar la implementación que esta página crea para usted.
9. Si no ha configurado previamente el rol de servicio de Greengrass en esta región, la consola abre un cuadro para configurar el rol de servicio por usted. El componente de autenticación del dispositivo de cliente usa este rol de servicio para verificar la identidad de los dispositivos de cliente y el componente detector de IP usa este rol de servicio para administrar la información de conectividad del dispositivo principal. Elija Conceder permisos.
10. En la página de Revisión, elija Implementar para iniciar la implementación en el dispositivo principal.
11. Para comprobar que la implementación se ha realizado correctamente, compruebe el estado de la implementación y compruebe los registros del dispositivo principal. Para comprobar el estado de la implementación en el dispositivo principal, puede elegir Objetivo en la Descripción general de la implementación. Para obtener más información, consulte los siguientes temas:

    - Comprobación del estado de la implementación
    - Supervisión de los registros de AWS IoT Greengrass

Paso 3: Conectar los dispositivos de cliente

Los dispositivos cliente pueden usarlo SDK para dispositivos con AWS IoT para detectar, conectarse y comunicarse con un dispositivo principal. El dispositivo cliente debe ser una AWS IoT cosa. Para obtener más información, consulte Crear un objeto en la Guía para desarrolladores de AWS IoT Core .

En esta sección, instalará la versión 2 del SDK para dispositivos con AWS IoT para Python y ejecutará la aplicación de ejemplo de detección de Greengrass desde el SDK para dispositivos con AWS IoT.

> **Note**
>
> También SDK para dispositivos con AWS IoT está disponible en otros lenguajes de programación. Este tutorial usa la SDK para dispositivos con AWS IoT v2 para Python, pero puedes explorar la otra SDKs para tu caso de uso. Para obtener más información, consulte AWS IoT Dispositivo SDKs en la guía para AWS IoT Core desarrolladores.

Conexión de un dispositivo de cliente a un dispositivo principal

1. Descargue e instale la SDK para dispositivos con AWS IoT versión 2 para Python en el AWS IoT dispositivo para conectarlo como dispositivo cliente.

   En el dispositivo de cliente, haga lo siguiente:

   a. Clona el repositorio SDK para dispositivos con AWS IoT v2 para Python para descargarlo.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Instale la SDK para dispositivos con AWS IoT versión 2 para Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Cambie a la carpeta de muestras de la SDK para dispositivos con AWS IoT versión 2 para Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples/greengrass
   ```

3. Ejecute la aplicación de ejemplo de detección de Greengrass. Esta aplicación espera argumentos que especifican el nombre del objeto del dispositivo de cliente, el tema y el mensaje de MQTT que se van a usar y los certificados que autentican y protegen la conexión. En el siguiente ejemplo, se envía el mensaje “Hello World” al tema `clients/`*`MyClientDevice1`*`/hello/world`.

   > **Note**
   >
   > Este tema coincide con el tema en el que configuró el puente MQTT para retransmitir mensajes al AWS IoT Core anterior.

   - *`MyClientDevice1`*Sustitúyalo por el nombre del dispositivo cliente.
   - *`~/certs/AmazonRootCA1.pem`*Sustitúyalo por la ruta al certificado de CA raíz de Amazon en el dispositivo cliente.
   - *`~/certs/device.pem.crt`*Sustitúyalo por la ruta al certificado del dispositivo del dispositivo cliente.
   - *`~/certs/private.pem.key`*Sustitúyalo por la ruta al archivo de clave privada del dispositivo cliente.

- *us-east-1*Sustitúyala por la AWS región en la que funcionan el dispositivo cliente y el dispositivo principal.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

La aplicación de ejemplo de detección envía el mensaje 10 veces y se desconecta. También se suscribe al mismo tema en el que publica los mensajes. Si el resultado indica que la aplicación recibió mensajes MQTT sobre el tema, el dispositivo de cliente puede comunicarse correctamente con el dispositivo principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
```

```
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

También puede ver los registros de Greengrass en el dispositivo principal para comprobar si el dispositivo de cliente se conecta y envía mensajes correctamente. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

4. Compruebe que el puente MQTT retransmita los mensajes del dispositivo cliente a. AWS IoT Core Puede utilizar el cliente de pruebas de MQTT de la AWS IoT Core consola para suscribirse a un filtro de temas de MQTT. Haga lo siguiente:

   a. Vaya a la consola de AWS IoT.

   b. En el panel de navegación izquierdo, en Prueba, elija Cliente de prueba MQTT.

   c. En la pestaña Suscribirse a un tema, en Filtro por tema, escriba `clients/+/hello/world` para suscribirse a los mensajes del dispositivo de cliente desde el dispositivo principal.

   d. Seleccione Suscribirse.

   e. Vuelva a ejecutar la publish/subscribe aplicación en el dispositivo cliente.

      El cliente de prueba MQTT muestra los mensajes que se envían desde el dispositivo de cliente sobre temas que coinciden con este filtro de temas.

Paso 4: Desarrollar un componente que se comunique con los dispositivos de cliente

Puede desarrollar componentes de Greengrass que se comuniquen con los dispositivos de cliente. Los componentes utilizan la comunicación entre procesos (IPC) y la publish/subscribe interfaz local para comunicarse en un dispositivo central. Para interactuar con los dispositivos cliente, configure el

componente de puente MQTT para retransmitir mensajes entre los dispositivos cliente y la interfaz local. publish/subscribe

En esta sección, actualizará el componente de puente MQTT para retransmitir mensajes desde los dispositivos cliente a la interfaz local publish/subscribe . A continuación, desarrolla un componente que se suscribe a estos mensajes y los imprime cuando los recibe.

Desarrollo de un componente que se comunique con los dispositivos de cliente

1. Revise la implementación en el dispositivo principal y configure el componente puente de MQTT para retransmitir los mensajes desde los dispositivos de cliente a la publicación/suscripción local. Haga lo siguiente:

   a. Vaya a la consola de AWS IoT Greengrass.

   b. En el panel de navegación de la izquierda, elija Dispositivos principales.

   c. En la página de Dispositivos principales, elija el dispositivo principal que va a usar para este tutorial.

   d. En la página de detalles del dispositivo principal, elija la pestaña Dispositivos de cliente.

   e. En la pestaña Dispositivos de cliente, elija Configurar la detección en la nube.

      Se abre la página Configurar la detección de dispositivos principales. En esta página, puede cambiar o configurar qué componentes del dispositivo de cliente se implementan en el dispositivo principal.

   f. En el Paso 3, para el componente aws.greengrass.clientdevices.mqtt.Bridge, elija Editar configuración.

   g. En el modal Editar configuración del componente de puente MQTT, configure un mapeo de temas que transmita los mensajes MQTT desde los dispositivos cliente a la interfaz local. publish/subscribe Haga lo siguiente:

      i. En Configuración, en bloque de códigos Configuración para combinar, introduzca la siguiente configuración. Esta configuración especifica la retransmisión de mensajes MQTT sobre temas que coincidan con el filtro de `clients/+/hello/world` temas desde los dispositivos cliente al servicio AWS IoT Core en la nube y al broker local de publish/subscribe Greengrass.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
```

```
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

Para obtener más información, consulte la configuración del componente puente de MQTT.

ii. Elija Confirmar.

h. Elija Revisar e implementar para revisar la implementación que esta página crea para usted.

i. En la página de Revisión, elija Implementar para iniciar la implementación en el dispositivo principal.

j. Para comprobar que la implementación se ha realizado correctamente, compruebe el estado de la implementación y compruebe los registros del dispositivo principal. Para comprobar el estado de la implementación en el dispositivo principal, puede elegir Objetivo en la Descripción general de la implementación. Para obtener más información, consulte los siguientes temas:

- Comprobación del estado de la implementación
- Supervisión de los registros de AWS IoT Greengrass

2. Desarrolle e implemente un componente de Greengrass que se suscriba a los mensajes de “Hello World” desde los dispositivos de cliente. Haga lo siguiente:

a. Cree carpetas para recetas y artefactos en el dispositivo principal.

Linux or Unix

```
mkdir recipes
mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

PowerShell

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

> **Important**
>
> Debe usar el siguiente formato para la ruta de la carpeta de artefactos. Incluya el nombre y la versión del componente que especifique en la receta.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. Use un editor de texto para crear una receta de componente con los siguientes contenidos. Esta receta especifica instalar la SDK para dispositivos con AWS IoT versión 2 para Python y ejecutar un script que se suscriba al tema e imprima los mensajes.

Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
```

Copie la siguiente receta en el archivo.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to Hello World messages
 from client devices.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
```

```
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    },
    "Manifests": [
      {
        "Platform": {
          "os": "linux"
        },
        "Lifecycle": {
          "install": "python3 -m pip install --user awsiotsdk",
          "Run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
        }
      },
      {
        "Platform": {
          "os": "windows"
        },
        "Lifecycle": {
          "install": "py -3 -m pip install --user awsiotsdk",
          "Run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
        }
      }
    ]
}
```

c. Use un editor de texto para crear un artefacto de script de Python denominado `hello_world_subscriber.py` con los siguientes contenidos. Esta aplicación utiliza el servicio publish/subscribe IPC para suscribirse al `clients/+/hello/world` tema e imprimir los mensajes que recibe.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

Copie el siguiente código de Python en el archivo.

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients/+/hello/world'
TIMEOUT = 10


def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
        traceback.print_exc()


try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
 on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')

    operation.close()
except Exception:
```

```
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

> **Note**
>
> Este componente usa el cliente IPC V2 en la SDK para dispositivos con AWS IoT v2 para Python para comunicarse con el software AWS IoT Greengrass Core. En comparación con el cliente IPC original, la versión 2 del cliente IPC reduce la cantidad de código que hay que escribir para usar el IPC en componentes personalizados.

d. Use la CLI de Greengrass para implementar el componente.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

3. Vea los registros de los componentes para comprobar que el componente se ha instalado correctamente y que está suscrito al tema.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MyHelloWorldSubscriber.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
Tail 10 -Wait
```

Puede mantener el feed de registro abierto para comprobar que el dispositivo principal recibe los mensajes.

4. En el dispositivo de cliente, vuelva a ejecutar la aplicación de ejemplo de detección de Greengrass para enviar mensajes al dispositivo principal.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

5. Vuelva a ver los registros de los componentes para comprobar que el componente recibe e imprime los mensajes del dispositivo de cliente.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MyHelloWorldSubscriber.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
Tail 10 -Wait
```

Paso 5: Desarrollar un componente que interactúe con las sombras de dispositivos de cliente

Puede desarrollar componentes de Greengrass que interactúen con las sombras de dispositivos de cliente de AWS IoT. Una sombra es un documento JSON que almacena la información de estado actual o deseada de una AWS IoT cosa, como un dispositivo cliente. Los componentes personalizados pueden acceder a las sombras de dispositivos de cliente para administrar su estado, incluso cuando el dispositivo de cliente no está conectado a AWS IoT. Cada AWS IoT elemento tiene una sombra sin nombre y también puede crear varias sombras con nombre para cada elemento.

En esta sección, implementará el componente administrador de sombras para administrar las sombras en el dispositivo principal. También actualice el componente puente de MQTT para retransmitir mensajes de sombra entre los dispositivos de cliente y el componente administrador de sombras. A continuación, desarrolle un componente que actualice las sombras de dispositivo de cliente y ejecute una aplicación de ejemplo en los dispositivos de cliente que responden a las actualizaciones de sombra del componente. Este componente representa una aplicación de administración de la iluminación inteligente, en la que el dispositivo principal administra el estado del color de las luces inteligentes que se conectan a él como dispositivos de cliente.

Desarrollo de un componente que interactúe con las sombras de dispositivos de cliente

1. Revise la implementación en el dispositivo principal para implementar el componente de administrador de sombras y configurar el componente puente de MQTT para retransmitir mensajes de sombra entre los dispositivos de cliente y la publicación/suscripción local, donde se comunica el administrador de sombras. Haga lo siguiente:

   a. Vaya a la consola de AWS IoT Greengrass.

   b. En el panel de navegación de la izquierda, elija Dispositivos principales.

   c. En la página de Dispositivos principales, elija el dispositivo principal que va a usar para este tutorial.

   d. En la página de detalles del dispositivo principal, elija la pestaña Dispositivos de cliente.

   e. En la pestaña Dispositivos de cliente, elija Configurar la detección en la nube.

      Se abre la página Configurar la detección de dispositivos principales. En esta página, puede cambiar o configurar qué componentes del dispositivo de cliente se implementan en el dispositivo principal.

f. En el Paso 3, para el componente aws.greengrass.clientdevices.mqtt.Bridge, elija Editar configuración.

g. En el cuadro Editar configuración del componente puente de MQTT, configure una asignación de temas que retransmita los mensajes MQTT en temas de sombra de dispositivo entre los dispositivos de cliente y la interfaz de publicación/suscripción local. También confirme que la implementación especifica una versión de puente de MQTT compatible. La compatibilidad con la sombra de dispositivo de cliente requiere la versión 2.2.0 o posterior del puente de MQTT. Haga lo siguiente:

   i. Para la Versión de componentes, elija la versión 2.2.0 o posterior.

   ii. En Configuración, en bloque de códigos Configuración para combinar, introduzca la siguiente configuración. Esta configuración especifica la retransmisión de mensajes MQTT sobre temas de sombra.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Para obtener más información, consulte la configuración del componente puente de MQTT.

iii. Elija Confirmar.

h. En el Paso 3, elija el componente aws.greengrass.ShadowManager para implementarlo.

i. Elija Revisar e implementar para revisar la implementación que esta página crea para usted.

j. En la página de Revisión, elija Implementar para iniciar la implementación en el dispositivo principal.

k. Para comprobar que la implementación se ha realizado correctamente, compruebe el estado de la implementación y compruebe los registros del dispositivo principal. Para comprobar el estado de la implementación en el dispositivo principal, puede elegir Objetivo en la Descripción general de la implementación. Para obtener más información, consulte los siguientes temas:

- Comprobación del estado de la implementación
- Supervisión de los registros de AWS IoT Greengrass

2. Desarrolle e implemente un componente de Greengrass que administre los dispositivos de cliente de iluminación inteligente. Haga lo siguiente:

a. Cree una carpeta con los artefactos del componente en el dispositivo principal.

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

> **Important**
>
> Debe usar el siguiente formato para la ruta de la carpeta de artefactos. Incluya el nombre y la versión del componente que especifique en la receta.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. Use un editor de texto para crear una receta de componente con los siguientes contenidos. Esta receta especifica instalar la SDK para dispositivos con AWS IoT versión 2 para Python y ejecutar un script que interactúe con las sombras de los dispositivos cliente de iluminación inteligente para gestionar sus colores.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
   ```

   Copie la siguiente receta en el archivo.

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.clientdevices.MySmartLightManager",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that interacts with smart light client
    devices.",
     "ComponentPublisher": "Amazon",
     "ComponentDependencies": {
       "aws.greengrass.Nucleus": {
         "VersionRequirement": "^2.6.0"
       },
       "aws.greengrass.ShadowManager": {
         "VersionRequirement": "^2.2.0"
       },
       "aws.greengrass.clientdevices.mqtt.Bridge": {
         "VersionRequirement": "^2.2.0"
       }
     },
     "ComponentConfiguration": {
       "DefaultConfiguration": {
   ```

```
      "smartLightDeviceNames": [],
      "accessControl": {
        "aws.greengrass.ShadowManager": {
          "com.example.clientdevices.MySmartLightManager:shadow:1": {
            "policyDescription": "Allows access to client devices' unnamed
shadows",
            "operations": [
              "aws.greengrass#GetThingShadow",
              "aws.greengrass#UpdateThingShadow"
            ],
            "resources": [
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        },
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MySmartLightManager:pubsub:1": {
            "policyDescription": "Allows access to client devices' unnamed
shadow updates",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "$aws/things/+/shadow/update/accepted"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "Run": "python3 -u {artifacts:path}/smart_light_manager.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
```

```
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "Run": "py -3 -u {artifacts:path}/smart_light_manager.py"
      }
    }
  ]
}
```

c. Use un editor de texto para crear un artefacto de script de Python denominado `smart_light_manager.py` con los siguientes contenidos. Esta aplicación utiliza el servicio IPC oculto para obtener y actualizar las sombras de los dispositivos cliente y el servicio publish/subscribe IPC local para recibir las actualizaciones ocultas notificadas.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py
```

   Copie el siguiente código de Python en el archivo.

```
import json
import random
import sys
import time
import traceback
from uuid import uuid4

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import ResourceNotFoundError

SHADOW_COLOR_PROPERTY = 'color'
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'
SET_COLOR_INTERVAL = 15


class SmartLightDevice():
    def __init__(self, client_device_name: str, reported_color: str = None):
        self.name = client_device_name
        self.reported_color = reported_color
```

```
        self.desired_color = None


class SmartLightDeviceManager():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):
        self.ipc_client = ipc_client
        self.devices = {}
        self.client_tokens = set()
        self.shadow_update_accepted_subscription_operation = None
        self.client_device_names_configuration_subscription_operation = None
        self.update_smart_light_device_list()

    def update_smart_light_device_list(self):
        # Update the device list from the component configuration.
        response = self.ipc_client.get_configuration(
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
        # Identify the difference between the configuration and the currently
 tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
 response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
 set(current_device_names)
        removed_device_names = set(current_device_names) -
 set(updated_device_names)
        # Stop tracking any smart light devices that are no longer in the
 configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)
        # Start tracking any new smart light devices that are in the
 configuration.
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
 device.reported_color))

    def get_device_reported_color(self, smart_light_device):
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
```

```
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color
                }
            },
            'clientToken': client_token
        }
        payload = bytes(json.dumps(payload_json), 'utf-8')
        self.ipc_client.update_thing_shadow(
            thing_name=smart_light_device.name, shadow_name='',
payload=payload)
        smart_light_device.desired_color = color

    def subscribe_to_shadow_update_accepted_events(self):
        if self.shadow_update_accepted_subscription_operation == None:
            # SubscribeToTopic returns a tuple with the response and the
operation.
            _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
                topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
            print('Successfully subscribed to shadow update accepted topic')

    def close_shadow_update_accepted_subscription(self):
        if self.shadow_update_accepted_subscription_operation is not None:
            self.shadow_update_accepted_subscription_operation.close()

    def on_shadow_update_accepted_event(self, event):
        try:
            message = str(event.binary_message.message, 'utf-8')
            accepted_payload = json.loads(message)
```

```
            # Check for reported states from smart light devices and ignore
desired states from components.
            if 'reported' in accepted_payload['state']:
                # Process this update only if it uses a client token created by
this component.
                client_token = accepted_payload.get('clientToken')
                if client_token is not None and client_token in
self.client_tokens:
                    self.client_tokens.remove(client_token)
                    shadow_state = accepted_payload['state']['reported']
                    if SHADOW_COLOR_PROPERTY in shadow_state:
                        reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                        topic = event.binary_message.context.topic
                        client_device_name = topic.split('/')[2]
                        if client_device_name in self.devices:
                            # Set the reported color for the smart light
device.
                            self.devices[client_device_name].reported_color =
reported_color
                            print(
                                'Received shadow update confirmation from
client device: %s' % client_device_name)
                    else:
                        print("Shadow update doesn't specify color")
        except:
            traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
            _, self.client_device_names_configuration_subscription_operation =
self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
light device names')

    def close_client_device_names_configuration_subscription(self):
        if self.client_device_names_configuration_subscription_operation is not
None:
```

```
 self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
 event.configuration_update_event.key_path:
                print('Received configuration update for list of client
 devices')
                self.update_smart_light_device_list()
        except:
            traceback.print_exc()


def choose_random_color():
    return random.choice(COLORS)

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

 smart_light_manager.subscribe_to_client_device_name_configuration_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                # Set each smart light device to a random color at a regular
 interval.
                for device_name in smart_light_manager.devices:
                    device = smart_light_manager.devices[device_name]
                    desired_color = choose_random_color()
                    print('Chose random color (%s) for %s' %
                          (desired_color, device_name))
                    if desired_color == device.desired_color:
                        print('Desired color for %s is already %s' %
                              (device_name, desired_color))
                    elif desired_color == device.reported_color:
                        print('Reported color for %s is already %s' %
                              (device_name, desired_color))
                    else:
                        smart_light_manager.request_device_color_change(
                            device, desired_color)
```

```
                        print('Requested color change for %s to %s' %
                              (device_name, desired_color))
                time.sleep(SET_COLOR_INTERVAL)
        except InterruptedError:
            print('Application interrupted')
        smart_light_manager.close_shadow_update_accepted_subscription()

 smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

Esta aplicación de Python hace lo siguiente:

- Lee la configuración del componente para obtener la lista de dispositivos de cliente de iluminación inteligente que hay que administrar.
- Se suscribe a las notificaciones de actualización de la configuración mediante la operación de IPC SubscribeToConfigurationUpdate. El software AWS IoT Greengrass Core envía notificaciones cada vez que cambia la configuración del componente. Cuando el componente recibe una notificación de actualización de la configuración, actualiza la lista de dispositivos cliente de iluminación inteligente que administra.
- Obtiene cada sombra de dispositivo de cliente de iluminación inteligente para obtener su estado de color inicial.
- Establece el color de cada dispositivo de cliente de iluminación inteligente en un color asignado al azar cada 15 segundos. El componente actualiza la sombra del objeto del dispositivo de cliente para cambiar su color. Esta operación envía un evento delta de sombra al dispositivo de cliente a través de MQTT.
- Se suscribe a la actualización de sombra de los mensajes aceptados en la interfaz de publicación/suscripción local mediante la operación de IPC SubscribeToTopic. Este componente recibe estos mensajes para rastrear el color de cada dispositivo de cliente de iluminación inteligente. Cuando un dispositivo de cliente de iluminación inteligente recibe una actualización de sombra, envía un mensaje MQTT para confirmar que ha recibido la actualización. El puente MQTT transmite este mensaje a la interfaz local publish/subscribe .

d. Use la CLI de Greengrass para implementar el componente. Al implementar este componente, se especifica la lista de dispositivos de cliente, `smartLightDeviceNames`, cuyas sombras administra. *MyClientDevice1*Sustitúyalo por el nombre del dispositivo cliente.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
  --update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
```

```
        ]
      }
    }
  }'
```

3. Vea los registros de los componentes para comprobar que el componente se instala y ejecuta correctamente.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
    10 -Wait
   ```

   El componente envía solicitudes para cambiar el color del dispositivo de cliente de iluminación inteligente. El administrador de sombras recibe la solicitud y establece el estado de la sombra `desired`. Sin embargo, el dispositivo de cliente de iluminación inteligente aún no está funcionando, por lo que el estado `reported` de la sombra no cambia. Los registros del componente incluyen los siguientes mensajes.

   ```
   2022-07-07T03:49:24.908Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
    for MyClientDevice1.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   2022-07-07T03:49:24.912Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout.
    Requested color change for MyClientDevice1 to blue.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   ```

   Puede mantener abierto el feed de registro abierto para ver cuándo imprime los mensajes el componente.

4. Descargue y ejecute una aplicación de ejemplo que use la detección de Greengrass y se suscribe a las actualizaciones de la sombra de dispositivo. En el dispositivo de cliente, haga lo siguiente:

   a. Use un editor de texto para crear un script de Python denominado `basic_discovery_shadow.py` con el siguiente contenido. Esta aplicación usa la detección y las sombras de Greengrass para mantener sincronizada una propiedad entre el dispositivo de cliente y el dispositivo principal.

      Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

      ```
      nano basic_discovery_shadow.py
      ```

      Copie el siguiente código de Python en el archivo.

      ```
      # Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
      # SPDX-License-Identifier: Apache-2.0.

      from awscrt import io
      from awscrt import mqtt
      from awsiot import iotshadow
      from awsiot.greengrass_discovery import DiscoveryClient
      from awsiot import mqtt_connection_builder
      from concurrent.futures import Future
      import sys
      import threading
      import traceback
      from uuid import uuid4

      # ------------------------------- ARGUMENT PARSING
       ---------------------------------------
      import argparse

      def parse_sample_input():
          parser = argparse.ArgumentParser(
              description="AWS IoT Greengrass Discovery Shadow",
              formatter_class=argparse.ArgumentDefaultsHelpFormatter,
          )

          # Connection / TLS
      ```

```
    parser.add_argument("--cert", required=True, dest="input_cert",
                        help="Path to the certificate file to use during mTLS
 connection establishment")
    parser.add_argument("--key", required=True, dest="input_key",
                        help="Path to the private key file to use during mTLS
 connection establishment")
    parser.add_argument("--ca_file", dest="input_ca", help="Path to optional CA
 bundle (PEM)")

    # Shadow
    parser.add_argument("--thing_name", required=True, dest="input_thing_name",
 help="The name assigned to your IoT Thing.")
    parser.add_argument("--shadow_property", dest="input_shadow_property",
 default="color",
                        help="The name of the shadow property you want to
 change (optional, default='color'")
    parser.add_argument("--region", dest="input_region", help="The region to
 connect through.", required=True)
    parser.add_argument("--print_discover_resp_only",
 dest="input_print_discover_resp_only", default=False)

    return parser.parse_args()

args = parse_sample_input()

# ---------------------------------- ARGUMENT PARSING END
 ----------------------------------------


# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
shadow_thing_name = args.input_thing_name
shadow_property = args.input_shadow_property

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()
```

```
locked_data = LockedData()


def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=args.input_cert,
                        pri_key_filepath=args.input_key,

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=args.input_thing_name,
                        clean_session=False,
                        keep_alive_secs=30)

                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection

                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
                    continue

    exit('All connection attempts failed')
```

```
# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
                return

        if response.state:
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
```

```
                    change_shadow_value(value)
                    return

            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
```

```
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
```

```
                else:
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value

def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return

        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
```

```
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
 topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
 None
        # for both reported and desired to clear the shadow document
 completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
 reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
                client_token=token,
            )

        future = shadow_client.publish_update_shadow(request,
 mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)


if __name__ == '__main__':
```

```
   tls_options =
io.TlsContextOptions.create_client_with_mtls_from_path(args.input_cert,args.input_key)
   if args.input_ca:
       tls_options.override_default_trust_store_from_path(None,
args.input_ca)
   tls_context = io.ClientTlsContext(tls_options)

   socket_options = io.SocketOptions()

   print('Performing greengrass discovery...')
   discovery_client =
DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
socket_options, tls_context, args.input_region)
   resp_future = discovery_client.discover(args.input_thing_name)
   discover_response = resp_future.result()

   print(discover_response)
   if args.print_discover_resp_only:
       exit(0)

   mqtt_connection = try_iot_endpoints()
   shadow_client = iotshadow.IotShadowClient(mqtt_connection)

   try:
       # Subscribe to necessary topics.
       # Note that is **is** important to wait for "accepted/rejected"
subscriptions
       # to succeed before publishing the corresponding "request".
       print("Subscribing to Update responses...")
       update_accepted_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_accepted(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
           qos=mqtt.QoS.AT_LEAST_ONCE,
           callback=on_update_shadow_accepted)

       update_rejected_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_rejected(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
           qos=mqtt.QoS.AT_LEAST_ONCE,
           callback=on_update_shadow_rejected)

       # Wait for subscriptions to succeed
```

```
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")

        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())
```

```
            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

Esta aplicación de Python hace lo siguiente:

- Usa la detección de Greengrass para descubrir el dispositivo principal y conectarse a él.
- Solicita el documento de sombra del dispositivo principal para obtener el estado inicial de la propiedad.
- Se suscribe a los eventos delta de sombra, que el dispositivo principal envía cuando el valor `desired` de la propiedad difiere de su valor `reported`. Cuando la aplicación recibe un evento delta de sombra, cambia el valor de la propiedad y envía una actualización al dispositivo principal para establecer el nuevo valor como su valor `reported`.

Esta aplicación combina la detección de Greengrass y las muestras de sombras de la v2 de SDK para dispositivos con AWS IoT .

b. Ejecute la aplicación de ejemplo. Esta aplicación espera argumentos que especifiquen el nombre del objeto del dispositivo de cliente, la propiedad de sombra que se va a usar y los certificados que autentican y protegen la conexión.

- *`MyClientDevice1`*Sustitúyalo por el nombre del dispositivo cliente.
- *`~/certs/AmazonRootCA1.pem`*Sustitúyalo por la ruta al certificado de CA raíz de Amazon en el dispositivo cliente.

- *~/certs/device.pem.crt*Sustitúyalo por la ruta al certificado del dispositivo del dispositivo cliente.
- *~/certs/private.pem.key*Sustitúyalo por la ruta al archivo de clave privada del dispositivo cliente.
- *us-east-1*Sustitúyala por la AWS región en la que funcionan el dispositivo cliente y el dispositivo principal.

```
python3 basic_discovery_shadow.py \
  --thing_name MyClientDevice1 \
  --shadow_property color \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbosity Warn
```

La aplicación de ejemplo se suscribe a los temas de sombra y espera a recibir los eventos delta de sombra del dispositivo principal. Si el resultado indica que la aplicación recibe eventos delta de sombra y responde a ellos, el dispositivo de cliente puede interactuar correctamente con su sombra en el dispositivo principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGG
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
Subscribing to Delta events...
Requesting current shadow state...
Received shadow delta event.
  Delta reports that desired value is 'purple'. Changing local value...
```

```
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

También puede ver los registros de Greengrass en el dispositivo principal para comprobar si el dispositivo de cliente se conecta y envía mensajes correctamente. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

5. Vuelva a ver los registros de los componentes para comprobar que el componente recibe confirmaciones de actualización de sombra del dispositivo de cliente de iluminación inteligente.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
    10 -Wait
   ```

   El componente registra los mensajes para confirmar que el dispositivo de cliente de iluminación inteligente ha cambiado de color.

   ```
   2022-07-07T03:49:24.908Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
    for MyClientDevice1.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   2022-07-07T03:49:24.912Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout.
    Requested color change for MyClientDevice1 to blue.
    {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
    serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
   2022-07-07T03:49:24.959Z [INFO] (Copier)
    com.example.clientdevices.MySmartLightManager: stdout. Received
   ```

```
shadow update confirmation from client device: MyClientDevice1.
{scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

> **Note**
>
> La sombra de dispositivo de cliente está sincronizada entre el dispositivo principal y el dispositivo de cliente. Sin embargo, el dispositivo principal no sincroniza la sombra del dispositivo cliente con AWS IoT Core. Por ejemplo, puedes sincronizar una sombra AWS IoT Core para ver o modificar el estado de todos los dispositivos de tu flota. Para obtener más información sobre cómo configurar el componente de administrador de sombras con el que se sincronizan las sombras AWS IoT Core, consulteSincronice las sombras de los dispositivos locales con AWS IoT Core.

Completó este tutorial. El dispositivo cliente se conecta al dispositivo principal, envía mensajes MQTT a los componentes de Greengrass AWS IoT Core y recibe actualizaciones instantáneas del dispositivo principal. Para obtener más información sobre los temas abordados en este tutorial, consulte lo siguiente:

- Asociación de los dispositivos de cliente
- Administración de puntos de conexión del dispositivo principal
- Prueba de las comunicaciones del dispositivo de cliente
- API de descubrimiento de Greengrass RESTful
- Retransmitir mensajes MQTT entre dispositivos cliente y AWS IoT Core
- Interacción con los dispositivos de cliente en los componentes
- Interacción con las sombras de dispositivo
- Interacción y sincronización con las sombras de dispositivo de cliente

Tutorial: Proteja Greengrass Nucleus con Trusted Platform Module (TPM)

> **Note**
>
> El mecanismo de este tutorial solo es compatible con Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.

Estos tutoriales contienen instrucciones sobre cómo usar el TPM2 chip como módulo de seguridad de hardware (HSM) para crear una clave privada y una CSR. El cual se utiliza para Creación del certificado del objeto.

Estos tutoriales le muestran cómo mejorar la seguridad de los dispositivos mediante la configuración del software AWS IoT Greengrass principal (Greengrass Nucleus) con un módulo de plataforma segura (TPM) mediante la interfaz PKCS #11. Esta integración del TPM garantiza que las claves privadas y los certificados utilizados para identificar los dispositivos y conectarse a ellos AWS IoT Core se almacenen de forma segura en un hardware a prueba de manipulaciones, lo que evita su extracción con fines de suplantación de identidad u otras actividades maliciosas.

Cuando complete esta integración, su dispositivo principal de Greengrass utilizará claves privadas protegidas por TPM para su identidad y comunicación con los servicios. AWS IoT

Para obtener más información acerca de la seguridad de los dispositivos de Greengrass, consulte Seguridad en AWS IoT Greengrass.

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Un dispositivo compatible con Linux con hardware TPM 2.0 o firmware TPM 2.0.
- Las instrucciones de este tutorial están definidas para Ubuntu 24.04 LTS.
  - Cualquier distribución de Linux que sea compatible con la pila de TPM2 software de Linux puede admitir este mecanismo.
- Una máquina de desarrollador con permisos AWS CLI instalados y configurados para:
  - Crear y administrar AWS IoT recursos

  - Cree y actualice los roles y las políticas de IAM
- La versión 8 o posterior del Entorno de ejecución de Java (JRE) instalado en su dispositivo.
- Los siguientes paquetes de software instalados en su dispositivo:
  - curl
  - jq
- Privilegios de raíz o sudo en el dispositivo.

Paso 1: Instalar TPM2 herramientas y dependencias

En este paso, instalará las herramientas y bibliotecas de TPM2 software necesarias.

1. Actualice su administrador de paquetes e instale las TPM2 herramientas y las dependencias ejecutando el siguiente comando.

```
sudo apt-get update && sudo apt-get install tpm2-tools \
tpm2-abrmd \
tpm2-tss-engine-tools \
gnutls-bin \
libtpm2-pkcs11-1 \
libtpm2-pkcs11-tools \
libtpm2-pkcs11-1-dev \
python3-tpm2-pkcs11-tools \
libengine-pkcs11-openssl \
libtss2-tcti-tabrmd0
```

2. Instale los paquetes del proveedor de TPM2 OpenSSL en Ubuntu 24.04 que utilizan el motor OpenSSL 3.

```
sudo apt-get install tpm2-openssl
```

Paso 2: inicializar el almacén PKCS#11 y crear un slot

1. Cree un directorio para almacenar datos.

```
sudo mkdir -p /etc/tpm2_pkcs11
```

2. Establezca la ubicación de almacenamiento como una variable de entorno. Para obtener más información sobre la jerarquía de almacenes, consulte Inicialización.

```
export TPM2_PKCS11_STORE=/etc/tpm2_store
```

3. Inicialice el token con el objeto principal TPM2 .

```
sudo tpm2_ptool init
```

Las siguientes opciones están disponibles:

`hierarchy-auth HIERARCHY_AUTH`

La contraseña de autorización para agregar un objeto principal a la jerarquía.

`primary-auth PRIMARY_AUTH`

Valor de autorización para el objeto de clave principal existente.

El valor predeterminado es un valor de autenticación vacío.

`primary-handle [PRIMARY_HANDLE]`

Utilice un objeto de clave principal existente.

Valor predeterminado: `0x81000001`

`transient-parents`

Utilice un objeto principal transitorio de una plantilla determinada.

Valores:`tpm2-tools-default`,, `tpm2-tools-ecc-default` `tss2-engine-key`

`path PATH`

La ubicación del directorio del almacén. Si se especifica, el directorio debe existir. Pero si no, realiza una búsqueda al observar la variable de entorno `TPM2_PKCS11_STORE`. Si esa

variable de entorno no está establecida, analizará `/etc/tpm2_pkcs11`. Si ese directorio no se encuentra o no se puede crear, por defecto será el directorio de trabajo actual.

Paso 3: crear un token y una clave

1. Cree un token PKCS#11.

```
sudo tpm2_ptool addtoken –pid=1 –userpin=USERPIN –sopin=SOPIN –label=greengrass
```

Las siguientes opciones están disponibles:

`--pid PID`

El identificador del objeto principal que se va a asociar a este token.

`--sopin SOPIN`

El pin del administrador. Este pin se utiliza para recuperar objetos.

`--userpin USERPIN`

El pin de usuario. Este PIN se utiliza para autenticar el uso de objetos.

`--label LABEL`

Una etiqueta única para identificar el perfil en uso. Debe ser única.

`--hierarchy-auth HIERARCHY_AUTH`

2. Cree un objeto clave ECC.

```
sudo tpm2_ptool addkey –algorithm=ecc256 –label=greengrass –userpin=****** –key-
label=greenkey
```

`--label LABEL`

La etiqueta del token también sirve para importar la clave.

`--key-label KEY_LABEL`

La etiqueta de la clave importada. El valor predeterminado es un número entero.

`--id ID`

ID de la clave. El valor predeterminado es un hexadecimal aleatorio de 8 bytes.

`--attr-always-authenticate`

Establece el atributo CKA_ALWAYS_AUTHENTICATE en CK_TRUE.

`--hierarchy-auth HIERARCHY_AUTH`

La hierarchyauth, necesaria para los objetos transitorios.

`--sopin SOPIN`

El pin de administrador.

`--userpin USERPIN`

El pin de usuario.

`--algorithm`

{rsa1024,rsa2048,rsa3072,rsa4096,aes128,aes256,ecc224,ecc256,ecc384,ecc521,hmac:sha1,hmac:s

El tipo de la clave.

3. Exporte el objeto TPM2 -TSS del token para capturar los datos de autenticación.

```
yaml_ecc0=$(sudo tpm2_ptool export –label="greengrass" –key-label="greenkey" –
userpin="******")
```

Ejemplo de código de salida:

```
> echo $yaml_ecc0
object-auth: 706c1cad8a5238871b30149705255926
primary-object:
  auth: ''
  hierarchy: owner
  is-transient: false
```

> **Note**
>
> También encontrarás un archivo llamado `greenkey.pem`, que es la clave TSS2 privada, en el directorio donde ejecutaste este comando. Utilícelo para generar la CSR con el proveedor tpm2 openssl. El archivo de clave TSS2 PRIVADA está protegido por el TPM y no se puede utilizar en otro equipo. Para obtener más información sobre TSS2 las claves con OpenSSL, consulte Almacenamiento de la clave pública o privada.

4. Capture los datos de autenticación de la clave privada del TSS.

```
auth_ecc0=$(echo "$yaml_ecc0" | grep "object-auth" | cut -d' ' -f2-)
```

Paso 4: genere una solicitud de firma de certificado (CSR)

En este paso, utilizarás la clave privada TPM2 protegida para generar una CSR.

1. Genera una CSR con el proveedor. TPM2

```
sudo openssl req -new -provider tpm2 -provider base -key greenkey.pem -passin "pass:
$auth_ecc0" -out "$H"$HOSTNAME".csr
```

Cuando se solicite, entregue la información necesaria para su CSR, como lo siguiente:

- Nombre del país (código de 2 letras)
- State or Province Name
- Locality Name
- Organization Name
- Organizational Unit Name
- Common Name
- Email Address

Puede otorgar, como alternativa, un archivo de configuración de OpenSSL para la generación desatendida. Para obtener más información, consulte la Documentación OpenSSL.

2. Si no va a generar la CSR en la misma máquina, copie la CSR generada en una máquina que tenga AWS las credenciales configuradas.

Paso 5: crear el certificado de objetos

Crea un certificado de AWS IoT cosas. Para obtener más información acerca de cómo crear un certificado de objetos, consulte Creación del certificado del objeto.

Paso 6: importar el certificado de objetos al TPM

1. Copie el certificado de objetos en el dispositivo.
2. Agregue el certificado de objetos al token de Greengrass.

```
sudo tpm2_ptool addcert --label=greengrass --key-label=greenkey device.pem.crt
```

Las siguientes opciones están disponibles:

`--help`

Muestre este mensaje de ayuda y salga de él.

`--label LABEL`

La etiqueta de perfil que se va a eliminar.

`--key-label KEY_LABEL`

La etiqueta de clave privada asociada.

`--key-id KEY_ID`

El ID de clave privada asociado en hexadecimal.

`cert`

El certificado x509 PEM que se va a agregar.

Paso 7: capturar la URL del objeto PKCS#11

Usaremos lo `p11tool` proporcionado en el `gnutls-bin` paquete para obtener la URL y el objeto URLs del token PKCS #11.

1. Capture la URL del token de Greengrass.

```
TOKEN=sudo p11tool --list-token-urls | grep "token=greengrass"
```

2. Consigue el objeto URLs para la ficha de Greengrass. Utilice el mismo pin que utilizó en el paso 3.

```
sudo p11tool --login --list-all "${TOKEN}"
```

   Ejemplo de código de salida:

```
Token 'greengrass' with URL
 'pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000
 requires user PIN
Enter PIN:
WARNING: Needed CKA_VALUE but didn't find encrypted blob
Object 0:
        URL:
 pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=00000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=private
        Type: Private key (EC/ECDSA-SECP256R1)
        Label: greenkey
        Flags: CKA_PRIVATE; CKA_NEVER_EXTRACTABLE; CKA_SENSITIVE;
        ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 1:
        URL:
 pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=00000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=public
        Type: Public key (EC/ECDSA-SECP256R1)
        Label: greenkey
        ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31

Object 2:
        URL:
 pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=00000000000000
%39%32%37%65%61%64%61%39%31%32%35%31%61%35%37%31;object=greenkey;type=cert
        Type: X.509 Certificate (EC/ECDSA-SECP256R1)
        Expires: Fri Dec 31 18:59:59 2049
        Label: greenkey
        ID: 39:32:37:65:61:64:61:39:31:32:35:31:61:35:37:31
```

3. Capture la URL del objeto para la clave privada y el certificado.

Paso 8: Configurar e instalar Greengrass con soporte TPM2

1. Configure el certificado de objetos. Para obtener más información, consulte Configurar el certificado de objetos.
2. Complete las instrucciones para instalar el software AWS IoT Greengrass principal con la clave privada y el certificado en un HSM in. Instale el software principal AWS IoT Greengrass A continuación, siga los siguientes pasos para configurar la instalación para que se aproveche a TPM2 través de la interfaz PKCS #11.
3. Compruebe que ha descargado y guardado el componente del proveedor PKCS#11 en la ubicación del instalador de Greengrass.
4. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador. Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU `nano` a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

5. Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema, los parámetros del núcleo de Greengrass y los parámetros del proveedor PKCS#11.

   ```
   ---
   system:
     certificateFilePath: "pkcs11:model=SW%20%20%20TPM
   %00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
   %34%35;object=greenkey;type=cert"
     privateKeyPath: "pkcs11:model=SW%20%20%20TPM
   %00%00%00%00%00%00%00%00;manufacturer=IBM;serial=0000000000000000;token=greengrass;id=
   %34%35;object=greenkey;type=private"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "myThing"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.14.0"
       configuration:
         awsRegion: "us-east-1"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   ```

```
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "tpm2_pkcs11"
      library: "/usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so"
      slot: 1
      userPin: "123456"
```

6. Edite el archivo con los parámetros específicos de su instalación.
   a. Actualice certificateFilePath y privateKeyPath con la actualización certificateFilePath y privateKeyPath con la URL de PKCS #11 capturada en el paso 7.
   b. Actualice su iotDataEndpoint anuncio iotCredEndpoint en función de su AWS IoT punto final.
   c. En la configuración `aws.greengrass.crypto.Pkcs11Provider`, actualice la biblioteca según su plataforma.

   > **Note**
   >
   > El ejemplo que se muestra es para X86_64. La ruta del archivo será similar para el ARM64 dispositivo.

7. Complete los pasos de instalación de Greengrass en Instale el software principal AWS IoT Greengrass.

Paso 9: verificar las instalaciones

En este paso, verificará que Greengrass funciona correctamente con TPM2 la integración.

1. Revise el estado del servicio de Greengrass.

```
sudo systemctl status greengrass.service
```

2. Consulte los registros de Greengrass para asegurarse de que no haya errores.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

3. Compruebe que el dispositivo aparece como conectado en la consola de AWS IoT.
   a. Inicie sesión en la consola de AWS IoT Greengrass.

b. En Manage (Administrar), expanda Greengrass devices (Dispositivos de Greengrass) y seleccione Core devices (Dispositivos principales).

c. Confirme que el dispositivo esté conectado. El estado del dispositivo se mostrará como `HEALTHY` si está conectado. Para obtener más información, consulte Comprobación del estado del dispositivo principal de Greengrass.

Resolución de problemas

Si encuentra problemas durante la configuración o el funcionamiento de su dispositivo Greengrass TPM2 habilitado, pruebe los siguientes pasos de solución de problemas.

- Revise el archivo de registro principal de Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

- Revise la configuración del proveedor PKCS#11.

```
sudo cat /greengrass/v2/config/effectiveConfig.yaml
```

- Asegúrese de que el TPM2 servicio esté funcionando.

```
sudo systemctl status tpm2-abrmd.service
```

- Compruebe que se pueda acceder a la TPM2 clave.

```
sudo pkcs11-tool –module /usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so -l -p
 123456 –list-objects
```

- Si su sistema operativo está configurado con un cifrado de disco completo con claves de raíz de TPM2 almacenamiento, como Clevis o systemd-cryptenroll, compruebe que no está utilizando el mismo identificador persistente que utilizan estas herramientas. El uso del mismo identificador persistente puede afectar al mecanismo de cifrado del disco. Ejecute el siguiente comando para comprobar todos los identificadores persistentes creados y utilizados

```
sudo tpm2_getcap handles-persistent
```

Siguientes pasos

Ahora que ha integrado correctamente su dispositivo principal de Greengrass TPM2, puede:

- Implementar componentes en su dispositivo de Greengrass seguro Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos
- Configure dispositivos Greengrass adicionales con TPM2 integración.

Para obtener más información acerca de la seguridad de los dispositivos de Greengrass, consulte Seguridad en AWS IoT Greengrass.

Tutorial: Proteja Nucleus Lite de AWS IoT Greengrass con Trusted Platform Module (TPM)

En este tutorial se explica cómo habilitar y configurar la compatibilidad con el módulo de plataforma segura (TPM) para AWS IoT Greengrass nucleus lite. El TPM proporciona una raíz de confianza basada en hardware para el almacenamiento seguro de claves. Esta función de seguridad protege las operaciones criptográficas y las credenciales confidenciales, lo que mejora la seguridad e integridad de los dispositivos.

Cuando complete esta integración, su dispositivo AWS IoT Greengrass principal utilizará claves privadas protegidas por TPM para su identidad y comunicación con los servicios. AWS IoT

Para obtener más información sobre la seguridad de AWS IoT Greengrass los dispositivos, consulte. Seguridad en AWS IoT Greengrass

> **Important**
>
> Este mecanismo solo es compatible con la instalación de AWS IoT Greengrass nucleus lite con aprovisionamiento manual de recursos.

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Un dispositivo compatible con Linux con hardware TPM 2.0 o NitroTPM

- Una máquina para desarrolladores con Greengrass Nucleus Lite instalado. Para obtener más información, consulte Instalación del software AWS IoT Greengrass principal (consola).
- Las instrucciones de este tutorial están definidas para Ubuntu 24.04 LTS.
- Cualquier distribución de Linux que sea compatible con la pila de TPM2 software de Linux puede admitir este mecanismo.
- Una máquina de desarrollador con la AWS CLI instalada y configurada con permisos para que:
  - Cree y administre AWS IoT recursos
  - Cree y actualice los roles y las políticas de IAM
- Privilegios de raíz o sudo en el dispositivo.

Este tutorial contiene instrucciones sobre cómo utilizar el TPM2 chip como módulo de seguridad de hardware (HSM) para crear una clave privada y la CSR, que se utiliza para crear el certificado del objeto AWS IoT .

Paso 1: Configurar una instancia de NitroTPM

1. Configura una instancia de NitroTPM. Para obtener más información, consulte NitroTPM.
2. Lance la última instancia con la AMI personalizada creada en el paso anterior.

   > **Important**
   >
   > Cuando se conecte con SSH, utilice el `ubuntu` usuario en lugar de`root`.

3. Compruebe que el dispositivo TPM está presente y funciona ejecutando el siguiente comando:

   ```
   ls -la /dev/tpm*
   ```

   Deberías ver los `/dev/tpm0` `/dev/tpmrm0` dispositivos.

Paso 2: Instalar y configurar las herramientas de TPM

1. Instale los paquetes necesarios ejecutando el siguiente comando:

   ```
   sudo apt update
   sudo apt install tpm2-openssl tpm2-tools tpm2-abrmd libtss2-tcti-tabrmd0
   ```

2. Verifique los permisos del dispositivo TPM ejecutando el siguiente comando:

```
ls -l /dev/tpm0    # Should be owned by tss:root with permissions 0660
ls -l /dev/tpmrm0  # Should be owned by tss:tss with permissions 0660
```

Paso 3: Configurar el proveedor de OpenSSL TPM2

1. Edite el archivo de configuración de OpenSSL:

```
sudo vi /etc/ssl/openssl.cnf
```

2. Agregue la siguiente configuración:

```
[openssl_init]
providers = provider_sect

[provider_sect]
default = default_sect
tpm2 = tpm2_sect

[default_sect]
activate = 1

[tpm2_sect]
identity = tpm2
module = /usr/local/lib64/tpm2.so
activate = 1
```

3. Ajuste la ruta del módulo según sea necesario. Para encontrar la ruta correcta, utilice:

```
find /usr -name "tpm2.so"
```

Paso 4: generar claves TPM persistentes

1. Cree una clave principal ejecutando el siguiente comando:

```
sudo tpm2_createprimary -C o -c primary.ctx
```

2. Cree un objeto de clave ECC ejecutando el siguiente comando:

```
sudo tpm2_create -C primary.ctx -g sha256 -G ecc256 -r device.priv -u device.pub
```

3. Cargue la clave ejecutando el siguiente comando:

```
sudo tpm2_load -C primary.ctx -r device.priv -u device.pub -c device.ctx
```

4. Haz que la clave sea persistente ejecutando el siguiente comando:

```
sudo tpm2_evictcontrol -C o -c device.ctx 0x81000002
```

   Esto crea una clave persistente con el identificador (tipo`0x81000002`).

Paso 5: generar una solicitud de firma de certificado (CSR)

En este paso, utilizará la clave privada TPM2 protegida para generar una solicitud de firma de certificado (CSR).

1. Genere una CSR con la clave TPM:

```
openssl req -new -provider tpm2 -key "handle:0x81000002" \
    -out device.csr \
    -subj "/CN=TPMThing"
```

2. `0x81000002`Sustitúyala por el valor de identificador que hayas elegido y por el `TPMThing` nombre de la cosa que desees.

Paso 6: Crea el certificado a partir de CSR

1. En tu ordenador de desarrollo, crea una carpeta donde descargaste el certificado de la AWS IoT cosa.

```
mkdir greengrass-v2-certs
```

2. Utilice el archivo CSR para crear y descargar el certificado del dispositivo en AWS IoT su ordenador de desarrollo.

```
aws iot create-certificate-from-csr \
  --set-as-active \
```

```
  --certificate-signing-request file://path_to_device.csr \
  --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

Si la solicitud se realiza correctamente, la respuesta tiene un aspecto similar al del siguiente ejemplo:

```
{
"certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Paso 7: Configurar Greengrass Nucleus Lite con compatibilidad con TPM

Para habilitar la compatibilidad con TPM en Nucleus Lite de Greengrass, realice los siguientes cambios:

1. Configure los permisos de usuario añadiendo el `ggcore` usuario al `tss` grupo para el acceso al TPM:

   ```
   sudo usermod -a -G tss ggcore
   ```

2. Actualice el directorio de credenciales siguiendo estos pasos:

- Elimine el archivo de clave privada del directorio de credenciales.
  - Como utilizamos claves TPM persistentes, no es necesario copiar ningún archivo de clave privada.
- Copie la nueva `device.pem.crt` en ese directorio de credenciales.

3. `config.yaml`Edítelo con la siguiente configuración específica de TPM:

```
system:
    privateKeyPath: "handle:0x81000002" # Use your chosen handle
    certificateFilePath: "" # Replace with the path of device.pem.crt
    ...
```

4. Reinicie su Greengrass Nucleus Lite ejecutando el siguiente comando:

```
systemctl restart greengrass-lite.target
```

Resolución de problemas

Si tiene problemas durante la configuración o el funcionamiento de su AWS IoT Greengrass dispositivo TPM2 compatible, pruebe los siguientes pasos de solución de problemas:

No se encuentra el dispositivo TPM (cuando se utiliza NitroTPM)

Si no `/dev/tpm0` está presente, lleve a cabo los siguientes pasos:

1. Comprueba que utilizas un tipo de instancia compatible con NitrotPM.
2. Asegúrese de que la AMI se haya creado con`--tpm-support v2.0`.
3. Compruebe que la instancia se lanzó desde la AMI personalizada.

Errores de permiso denegado

Si encuentra errores de acceso al TPM, haga lo siguiente:

1. Compruebe que el usuario está en el `tss` grupo:`groups $USER`.
2. Compruebe los permisos del dispositivo TPM mediante el siguiente comando:

```
ls -l /dev/tpm*
```

3. Compruebe que ha cerrado sesión y ha vuelto a iniciarla después de añadirla al `tss` grupo.

Problemas con el proveedor de OpenSSL

Si no encuentra el proveedor de TPM, haga lo siguiente:

1. Compruebe `tpm2.so` la ruta en. `/etc/ssl/openssl.cnf`
2. Compruebe la instalación del proveedor ejecutando el siguiente comando:

```
openssl list -providers
```

3. Compruebe que el `tpm2-openssl` paquete esté instalado correctamente.

Siguientes pasos

Ahora que has integrado correctamente tu dispositivo AWS IoT Greengrass principal TPM2, puedes:

- Implemente componentes en su AWS IoT Greengrass dispositivo seguro
- Configure AWS IoT Greengrass dispositivos adicionales con TPM2 la integración

Para obtener más información sobre la seguridad de AWS IoT Greengrass los dispositivos, consulte[Seguridad en AWS IoT Greengrass](#).

Tutorial: Cómo empezar a SageMaker usar AI Edge Manager

> **Important**
>
> SageMaker AI Edge Manager se suspendió el 26 de abril de 2024. Para obtener más información sobre cómo seguir implementando sus modelos en dispositivos periféricos, consulte el final del ciclo de vida de SageMaker AI Edge Manager.

Amazon SageMaker AI Edge Manager es un agente de software que se ejecuta en dispositivos periféricos. SageMaker AI Edge Manager proporciona administración de modelos para dispositivos periféricos para que pueda empaquetar y usar modelos compilados por Amazon SageMaker AI NEO directamente en los dispositivos principales de Greengrass. Con SageMaker AI Edge Manager, también puede muestrear los datos de entrada y salida del modelo de sus dispositivos principales y

enviarlos a ellos Nube de AWS para su supervisión y análisis. Para obtener más información sobre cómo funciona SageMaker AI Edge Manager en los dispositivos principales de Greengrass, consulte. Utilice Amazon SageMaker AI Edge Manager en los dispositivos principales de Greengrass

En este tutorial, se muestra cómo empezar a utilizar SageMaker AI Edge Manager con los componentes AWS de muestra proporcionados en un dispositivo principal existente. Estos componentes de ejemplo utilizan el componente SageMaker AI Edge Manager como una dependencia para implementar el agente de Edge Manager y realizar inferencias utilizando modelos previamente entrenados que se compilaron con SageMaker AI Neo. Para obtener más información sobre el agente de SageMaker AI Edge Manager, consulte SageMaker AI Edge Manager en la Guía para desarrolladores de Amazon SageMaker AI.

Para configurar y usar el agente SageMaker AI Edge Manager en un dispositivo principal de Greengrass existente, AWS proporciona un código de ejemplo que puede usar para crear los siguientes ejemplos de componentes de inferencia y modelo.

- Clasificación de imágenes
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- Detección de objetos
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

En este tutorial, se muestra cómo implementar los componentes de muestra y el agente de SageMaker AI Edge Manager.

Temas

- Requisitos previos
- Configura tu dispositivo principal de Greengrass en SageMaker AI Edge Manager
- Creación de los componentes de muestra
- Ejecución de un ejemplo de inferencia de clasificación de imágenes

Requisitos previos

Para completar este tutorial, necesita cumplir con los siguientes requisitos previos:

- Un dispositivo principal de Greengrass que se ejecuta en Amazon Linux 2, una plataforma de Linux basada en Debian (x86_64 o Armv8) o Windows (x86_64). Si no dispone de una, consulte Tutorial: Introducción a AWS IoT Greengrass V2.
- Python 3.6 o posterior, incluido `pip` para la versión de Python, instalado en el dispositivo principal.
- El tiempo de ejecución de OpenGL API GLX (`libgl1-mesa-glx`) instalado en el dispositivo principal.
- Un usuario AWS Identity and Access Management (IAM) con permisos de administrador.
- Un equipo de desarrollo de tipo Windows, Mac o Unix con acceso a Internet y que cumpla los siguientes requisitos:
  - Versión 3.6 o posterior de Python instalada.
  - AWS CLI instalado y configurado con sus credenciales de usuario administrador de IAM. Para obtener más información, consulte Instalación de la AWS CLI y Configuración de la AWS CLI.
- Los siguientes depósitos de S3 se crearon en el mismo dispositivo principal de Greengrass Cuenta de AWS y en el Región de AWS mismo que él:
  - Un bucket de S3 para almacenar los artefactos que se incluyen en los componentes de inferencia y modelo de muestra. En este tutorial se utiliza amzn-s3-demo-bucket1 para referirse a este bucket.
  - Un bucket de S3 que asocie a su flota de dispositivos SageMaker periféricos de IA. SageMaker AI Edge Manager requiere un depósito S3 para crear la flota de dispositivos perimetrales y almacenar datos de muestra derivados de la ejecución de inferencias en tu dispositivo. En este tutorial se utiliza amzn-s3-demo-bucket2 para referirse a este bucket.

  Para más información sobre la creación de buckets de S3, consulte Introducción a Amazon S3.
- El rol del dispositivo de Greengrass se configuró con lo siguiente:
  - Una relación de confianza que permite a `credentials.iot.amazonaws.com` y a `sagemaker.amazonaws.com` asumir el rol, como se muestra en el siguiente ejemplo de política de IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
```

```
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- La política gestionada AmazonSageMakerEdgeDeviceFleetPolicyde IAM.
- La política gestionada por AmazonSageMakerFullAccessIAM.
- La acción `s3:GetObject` del bucket de S3 que contiene los artefactos de sus componentes, como se muestra en el siguiente ejemplo de política de IAM.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
  {
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket1/*"
    ],
    "Effect": "Allow"
  }
  ]
}
```

Configura tu dispositivo principal de Greengrass en SageMaker AI Edge Manager

Las flotas de dispositivos perimetrales de SageMaker AI Edge Manager son conjuntos de dispositivos agrupados de forma lógica. Para usar SageMaker AI Edge Manager con AI Edge Manager AWS IoT Greengrass, debe crear una flota de dispositivos perimetrales que utilice el mismo alias de AWS IoT

rol que el dispositivo principal de Greengrass en el que despliega el agente de SageMaker AI Edge Manager. A continuación, debe registrar el dispositivo principal como parte de esa flota.

Temas

- Creación de una flota de dispositivos de periferia
- Registro del dispositivo principal de Greengrass

Creación de una flota de dispositivos de periferia

Cómo crear una flota de dispositivos de periferia (consola)

1. En la consola Amazon SageMaker AI, selecciona Edge Manager y, a continuación, elige flotas de dispositivos Edge.
2. En la página Flotas de dispositivos, elija Crear una flota de dispositivos.
3. En Propiedades de la flota de dispositivos, haga lo siguiente:
   - En Nombre de la flota de dispositivos, ingrese un nombre para la flota de dispositivos.
   - Para el rol de IAM, ingrese el nombre de recurso de Amazon (ARN) del alias del rol AWS IoT que especificó cuando configuró el dispositivo principal de Greengrass.
   - Desactive la opción Crear alias de rol de IAM.
4. Elija Siguiente.
5. En Configuración de salida, para el URI del bucket de S3, introduzca el URI del bucket de S3 que desee asociar a la flota de dispositivos.
6. Seleccione Enviar.

Registro del dispositivo principal de Greengrass

Cómo registrar el dispositivo principal de Greengrass como dispositivo de periferia (consola)

1. En la consola Amazon SageMaker AI, selecciona Edge Manager y, a continuación, elige dispositivos Edge.
2. En la página Dispositivos, seleccione Registrar dispositivo.
3. Dentro de Propiedades del dispositivo, en Nombre de la flota de dispositivos, introduzca el nombre de la flota de dispositivos que creó y, a continuación, seleccione Siguiente.
4. Elija Siguiente.

5. En Fuente del dispositivo, en Nombre del dispositivo, introduzca el AWS IoT nombre del dispositivo principal de Greengrass.
6. Seleccione Enviar.

Creación de los componentes de muestra

Para ayudarle a empezar a utilizar el componente SageMaker AI Edge Manager, AWS proporciona un script de Python GitHub que crea los componentes de inferencia y modelo de muestra y los carga en su lugar Nube de AWS . Complete los siguientes pasos en una computadora de desarrollo.

Cómo crear los componentes de muestra

1. Descarga el repositorio de ejemplos de AWS IoT Greengrass componentes en tu GitHub ordenador de desarrollo.
2. Diríjase a la carpeta `/machine-learning/sagemaker-edge-manager` descargada.

   ```
   cd download-directory/machine-learning/sagemaker-edge-manager
   ```

3. Ejecute el siguiente comando para crear y cargar los componentes de muestra en la Nube de AWS.

   ```
   python3 create_components.py -r region -b amzn-s3-demo-bucket
   ```

   *region*Sustitúyalo por el Región de AWS lugar donde creó su dispositivo principal de Greengrass y sustituya amzn-s3-demo-bucket1 por el nombre del depósito S3 para almacenar los artefactos de sus componentes.

   > **Note**
   >
   > De forma predeterminada, el script crea componentes de muestra tanto para la clasificación de imágenes como para la inferencia de detección de objetos. Para crear componentes solo para un tipo específico de inferencia, especifique el argumento `-i` *`ImageClassification | ObjectDetection`*.

Los componentes de inferencia y modelo de muestra para usarlos con AI Edge Manager ahora están creados en su. SageMaker Cuenta de AWS Para ver los componentes de muestra en la consola

de AWS IoT Greengrass, elija Componentes y, a continuación, en Mis componentes, busque los siguientes componentes:

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Ejecución de un ejemplo de inferencia de clasificación de imágenes

Para realizar una inferencia de clasificación de imágenes con los componentes de muestra AWS proporcionados y el agente SageMaker AI Edge Manager, debes implementar estos componentes en tu dispositivo principal. Al implementar estos componentes, se descarga un modelo Resnet-50 prediseñado y compilado por SageMaker AI NEO e instala el agente AI Edge Manager en el SageMaker dispositivo. El agente SageMaker AI Edge Manager carga el modelo y publica los resultados de las inferencias sobre el tema. `gg/sageMakerEdgeManager/image-classification` Para ver estos resultados de inferencia, usa el cliente AWS IoT MQTT de la AWS IoT consola para suscribirte a este tema.

Temas

- Suscripción al tema de notificaciones
- Implementación de los componentes de muestra
- Visualización de los resultados de inferencia

Suscripción al tema de notificaciones

En este paso, configurará el cliente AWS IoT MQTT de la AWS IoT consola para ver los mensajes MQTT publicados por el componente de inferencia de muestra. De forma predeterminada, el componente publica los resultados de las inferencias sobre el tema `gg/sageMakerEdgeManager/image-classification`. Consulte este tema antes de implementar el componente en el dispositivo principal de Greengrass para ver los resultados de la inferencia cuando el componente se ejecute por primera vez.

Cómo suscribirse al tema de notificaciones predeterminado

1. En el menú de navegación de la consola de AWS IoT, seleccione Prueba, cliente de prueba de MQTT.
2. En Suscripción a un tema, en el cuadro Nombre del tema, introduzca **`gg/sageMakerEdgeManager/image-classification`**.
3. Seleccione Suscribirse.

Implementación de los componentes de muestra

En este paso, configurará e implementará los siguientes componentes en el dispositivo principal:

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

Para implementar sus componentes (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Implementaciones y, a continuación, elija la implementación del dispositivo de destino que desee revisar.
2. En la página de implementación, elija Revisar y, a continuación, elija Revisar implementación.
3. En la página Especificar destino, seleccione Siguiente.
4. En la página Seleccionar componentes, haga lo siguiente:
   a. En Mis componentes, seleccione los siguientes componentes:
      - `com.greengrass.SageMakerEdgeManager.ImageClassification`
      - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
   b. En Componentes públicos, desactive la opción Mostrar solo los componentes seleccionados y, a continuación, seleccione el componente `aws.greengrass.SageMakerEdgeManager`.
   c. Elija Siguiente.
5. En la página Configurar componentes, seleccione el componente `aws.greengrass.SageMakerEdgeManager` y haga lo siguiente.
   a. Seleccione Configurar componente.

b. En Actualización de la configuración, en Configuración para fusionar, introduzca la siguiente configuración.

```
{
    "DeviceFleetName": "device-fleet-name",
    "BucketName": "amzn-s3-demo-bucket"
}
```

*device-fleet-name*Sustitúyalo por el nombre de la flota de dispositivos periféricos que has creado y *amzn-s3-demo-bucket* sustitúyelo por el nombre del depósito de S3 asociado a tu flota de dispositivos.

c. Seleccione Confirmar y, a continuación, elija Siguiente.

6. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
7. En la página Revisar, elija Implementar.

Para implementar sus componentes (AWS CLI)

1. En su ordenador de desarrollo, cree un `deployment.json` archivo para definir la configuración de despliegue de los componentes de SageMaker AI Edge Manager. Este archivo debería ser igual al siguiente ejemplo.

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\":\"device-fleet-name\",\"BucketName\":\"amzn-
s3-demo-bucket2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
```

```
      "configurationUpdate": {
      }
    },
  }
}
```

- En el campo `targetArn`, sustituya *`targetArn`* por el nombre de recurso de Amazon (ARN) de la cosa o grupo de cosas a la que apunte la implementación, en el siguiente formato:
  - Cosa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
  - Grupo de cosas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
- En el `merge` campo, *`device-fleet-name`* sustitúyalo por el nombre de la flota de dispositivos perimetrales que has creado. A continuación, *`amzn-s3-demo-bucket2`* sustitúyalo por el nombre del depósito de S3 asociado a tu flota de dispositivos.
- Sustituya las versiones de cada componente por la última versión disponible.

2. Ejecute el siguiente comando para implementar los componentes en el dispositivo:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

La implementación puede tardar varios minutos en completarse. En el siguiente paso, compruebe el registro de componentes para comprobar que la implementación se ha completado correctamente y para ver los resultados de la inferencia.

Visualización de los resultados de inferencia

Tras implementar los componentes, puede ver los resultados de la inferencia en el registro de componentes de su dispositivo principal de Greengrass y en AWS IoT el cliente MQTT de la consola. AWS IoT Para suscribirse al tema sobre el que el componente publica los resultados de las inferencias, consulte Suscripción al tema de notificaciones.

- AWS IoT Cliente MQTT: para ver los resultados que el componente de inferencia publica en el tema de notificaciones predeterminado, complete los siguientes pasos:

  1. En el menú de navegación de la consola de AWS IoT, seleccione Prueba, cliente de prueba de MQTT.
  2. En Suscripciones, elija **`gg/sageMakerEdgeManager/image-classification`**.

- Registro de componentes: para ver los resultados de la inferencia en el registro de componentes, ejecute el siguiente comando en el dispositivo principal de Greengrass.

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

Si no puede ver los resultados de la inferencia en el registro de componentes o en el cliente MQTT, significa que la implementación falló o no llegó al dispositivo principal. Esto puede ocurrir si el dispositivo principal no está conectado a Internet o no tiene los permisos adecuados para ejecutar el componente. Ejecute el siguiente comando en su dispositivo principal para ver el archivo de registro del software AWS IoT Greengrass principal. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Para obtener más información, consulte Resolución de problemas de inferencia de machine learning.

Tutorial: Realice una inferencia de clasificación de imágenes de muestra con Lite TensorFlow

En este tutorial se muestra cómo utilizar el componente de inferencia de clasificación de imágenes de TensorFlow Lite para realizar una inferencia de clasificación de imágenes de muestra en un dispositivo central de Greengrass. Este componente incluye las siguientes dependencias de componentes:

- TensorFlow Componente de tienda de modelos de clasificación de imágenes Lite
- TensorFlow Componente de tiempo de ejecución Lite

Al implementar este componente, descarga un modelo MobileNet v1 previamente entrenado e instala el motor de ejecución de TensorFlow Lite y sus dependencias. Este componente publica los resultados de las inferencias en el tema `ml/tflite/image-classification`. Para ver estos resultados de inferencia, utilice el cliente AWS IoT MQTT de la AWS IoT consola para suscribirse a este tema.

En este tutorial, implementará el componente de inferencia de muestra para realizar la clasificación de imágenes en la imagen de muestra que proporciona AWS IoT Greengrass. Después de completar este tutorial, puede completar Tutorial: Realice una inferencia de clasificación de imágenes de muestra en imágenes de una cámara con Lite TensorFlow, que muestra cómo modificar el componente de inferencia de muestras para realizar la clasificación de imágenes en las imágenes de una cámara de forma local en un dispositivo principal de Greengrass.

Para obtener más información acerca de machine learning en dispositivos de Greengrass, consulte Cómo realizar la inferencia de machine learning.

Temas

- Requisitos previos
- Paso 1: Suscribirse al tema de notificaciones predeterminado
- Paso 2: Implemente el componente de clasificación de imágenes TensorFlow Lite
- Paso 3: Visualizar los resultados de la inferencia
- Siguientes pasos

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Un dispositivo principal de Greengrass para Linux. Si no dispone de una, consulte Tutorial: Introducción a AWS IoT Greengrass V2. El dispositivo principal debe cumplir los siguientes requisitos:
  - En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
  - En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

    ```
    sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
     libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
    ```

  - Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:

- NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

  ```
  pip3 install --upgrade numpy
  ```

- La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

  Cómo activar la pila de cámara antigua

  1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Seleccione Opciones de interfaz.
  3. Seleccione Cámara antigua para activar la pila de cámara antigua.
  4. Reinicie el Raspberry Pi.

Paso 1: Suscribirse al tema de notificaciones predeterminado

En este paso, debe configurar el cliente AWS IoT MQTT de la AWS IoT consola para ver los mensajes MQTT publicados por el componente de clasificación de imágenes de TensorFlow Lite. De forma predeterminada, el componente publica los resultados de las inferencias sobre el tema `ml/tflite/image-classification`. Consulte este tema antes de implementar el componente en el dispositivo principal de Greengrass para ver los resultados de la inferencia cuando el componente se ejecute por primera vez.

Cómo suscribirse al tema de notificaciones predeterminado

1. En el menú de navegación de la consola de AWS IoT, seleccione Prueba, cliente de prueba de MQTT.
2. En Suscripción a un tema, en el cuadro Nombre del tema, introduzca **`ml/tflite/image-classification`**.
3. Seleccione Suscribirse.

Paso 2: Implemente el componente de clasificación de imágenes TensorFlow Lite

En este paso, implementará el componente de clasificación de imágenes TensorFlow Lite en su dispositivo principal:

Para implementar el componente de clasificación de imágenes TensorFlow Lite (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, en la pestaña Componentes públicos, elija `aws.greengrass.TensorFlowLiteImageClassification`.
3. En la página aws.greengrass.TensorFlowLiteImageClassification, elija Implementar.
4. En Agregar a la implementación, elija una de las siguientes opciones:
   a. Para combinar este componente con una implementación existente en el dispositivo de destino, elija Agregar a la implementación existente y, a continuación, seleccione la implementación que desee revisar.
   b. Para crear una nueva implementación en el dispositivo de destino, elija Crear nueva implementación. Si tiene una implementación existente en su dispositivo, al elegir este paso se reemplaza la implementación existente.
5. En la página Especificar detalles, haga lo siguiente:
   a. En Información de implementación, introduzca o modifique el nombre descriptivo de su implementación.
   b. En Objetivos de implementación, seleccione un objetivo para su implementación y elija Siguiente. No puede cambiar el objetivo de implementación si está revisando una implementación existente.
6. En la página Seleccionar componentes, en Componentes públicos, compruebe que el componente `aws.greengrass.TensorFlowLiteImageClassification` esté seleccionado y elija Siguiente.
7. En la página Configurar componentes, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Revisar, elija Implementar.

Para implementar el componente de clasificación de imágenes TensorFlow Lite (AWS CLI)

1. Cree un `deployment.json` archivo para definir la configuración de despliegue del componente de clasificación de imágenes de TensorFlow Lite. Este archivo debería tener el siguiente aspecto:

   ```
   {
     "targetArn":"targetArn",
     "components": {
       "aws.greengrass.TensorFlowLiteImageClassification": {
         "componentVersion": 2.1.0,
         "configurationUpdate": {
         }
       }
     }
   }
   ```

   - En el campo `targetArn`, sustituya *`targetArn`* por el Nombre de recurso de Amazon (ARN) del objeto o grupo de objetos a la que apunte la implementación, en el siguiente formato:
     - Cosa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Grupo de cosas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
   - Este tutorial utiliza la versión 2.1.0 de componente. En el objeto `aws.greengrass.TensorFlowLiteObjectDetection` componente, *`2.1.0`* sustitúyalo por una versión diferente del componente de detección de objetos TensorFlow Lite.
2. Ejecute el siguiente comando para implementar el componente de clasificación de imágenes TensorFlow Lite en el dispositivo:

   ```
   aws greengrassv2 create-deployment \
       --cli-input-json file://path/to/deployment.json
   ```

La implementación puede tardar varios minutos en completarse. En el siguiente paso, compruebe el registro de componentes para comprobar que la implementación se ha completado correctamente y para ver los resultados de la inferencia.

Paso 3: Visualizar los resultados de la inferencia

Tras implementar el componente, puede ver los resultados de la inferencia en el registro del componente de su dispositivo principal de Greengrass y en AWS IoT el cliente MQTT de la consola.

AWS IoT Para suscribirse al tema sobre el que el componente publica los resultados de las inferencias, consulte Paso 1: Suscribirse al tema de notificaciones predeterminado.

- AWS IoT Cliente MQTT: para ver los resultados que el componente de inferencia publica en el tema de notificaciones predeterminado, complete los siguientes pasos:

  1. En el menú de navegación de la consola de AWS IoT, seleccione Prueba, cliente de prueba de MQTT.
  2. En Suscripciones, elija **`ml/tflite/image-classification`**.

     Debería ver mensajes similares al del siguiente ejemplo.

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
  "inference-type": "image-classification",
  "inference-description": "Top 5 predictions with score 0.3 or above ",
  "inference-results": [
    {
      "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "Persian cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "tiger cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "dalmatian, coach dog, carriage dog",
      "Score": "0.5607843137254902"
    },
    {
      "Label": "malamute, malemute, Alaskan malamute",
      "Score": "0.5450980392156862"
    }
  ]
}
```

- Registro de componentes: para ver los resultados de la inferencia en el registro de componentes, ejecute el siguiente comando en el dispositivo principal de Greengrass.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
  ```

  Debería ver resultados similares al del siguiente ejemplo.

  ```
  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
   IoT core....
   {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
   serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
   "2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-
  description": "Top 5 predictions with score 0.3 or above ", "inference-results":
   [{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
   concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":
   "0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},
   {"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},
   {"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]}.
   {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
   serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
  ```

Si no puede ver los resultados de la inferencia en el registro de componentes o en el cliente MQTT, significa que la implementación falló o no llegó al dispositivo principal. Esto puede ocurrir si el dispositivo principal no está conectado a Internet o no tiene los permisos adecuados para ejecutar el componente. Ejecute el siguiente comando en su dispositivo principal para ver el archivo de registro del software AWS IoT Greengrass principal. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Para obtener más información, consulte Resolución de problemas de inferencia de machine learning.

Siguientes pasos

Si tiene un dispositivo principal de Greengrass con una interfaz de cámara compatible, puede completar Tutorial: Realice una inferencia de clasificación de imágenes de muestra en imágenes de una cámara con Lite TensorFlow, que muestra cómo modificar el componente de inferencia de muestras para clasificar las imágenes de una cámara.

Para explorar más a fondo la configuración del componente de inferencia de clasificación de imágenes de TensorFlow Lite de muestra, pruebe lo siguiente:

- Modifique el parámetro de configuración `InferenceInterval` para cambiar la frecuencia con la que se ejecuta el código de inferencia.
- Modifique los parámetros de configuración `ImageName` y `ImageDirectory` en la configuración del componente de inferencia para especificar una imagen personalizada que se utilizará en la inferencia.

Para obtener más información sobre la personalización de la configuración de los componentes públicos o la creación de componentes de machine learning personalizados, consulte Personalización de sus componentes de machine learning.

Tutorial: Realice una inferencia de clasificación de imágenes de muestra en imágenes de una cámara con Lite TensorFlow

Este tutorial le muestra cómo utilizar el componente de inferencia de clasificación de imágenes TensorFlow Lite para realizar inferencias de clasificación de imágenes de muestra en imágenes de una cámara local en un dispositivo central de Greengrass. Este componente incluye las siguientes dependencias de componentes:

- TensorFlow Componente de tienda de modelos de clasificación de imágenes Lite
- TensorFlow Componente de tiempo de ejecución Lite

> **Note**
>
> Este tutorial permite acceder al módulo de cámara de los dispositivos Raspberry Pi o NVIDIA Jetson Nano, pero AWS IoT Greengrass es compatible con otros dispositivos en

las plataformas ARMv7L, Armv8 o x86_64. Para configurar una cámara para un dispositivo diferente, consulte la documentación correspondiente al dispositivo.

Para obtener más información acerca de machine learning en dispositivos de Greengrass, consulte Cómo realizar la inferencia de machine learning.

Temas

- Requisitos previos
- Paso 1: Configurar el módulo de cámara del dispositivo
- Paso 2: Comprobar la suscripción al tema de notificaciones predeterminado
- Paso 3: Modifique la configuración del componente de clasificación de imágenes de TensorFlow Lite e impleméntelo
- Paso 4: Visualizar los resultados de la inferencia
- Siguientes pasos

Requisitos previos

Para completar este tutorial, primero debe completar Tutorial: Realice una inferencia de clasificación de imágenes de muestra con Lite TensorFlow.

También necesitará lo siguiente:

- Un dispositivo principal de Linux Greengrass con una interfaz de cámara. Este tutorial permite acceder al módulo de cámara en uno de los siguientes dispositivos compatibles:
  - Raspberry Pi con el sistema operativo Raspberry Pi (anteriormente llamado Raspbian)
  - NVIDIA Jetson Nano

  Para obtener información acerca de cómo configurar un dispositivo principal de Greengrass, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

  El dispositivo principal debe cumplir los siguientes requisitos:

  - En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
  - En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.
- Para dispositivos Raspberry Pi o NVIDIA Jetson Nano, módulo de cámara Raspberry Pi V2, 8 megapíxeles, 1080p. Para obtener información sobre cómo configurar la cámara, consulte Connecting the camera en la documentación de Raspberry Pi.

Paso 1: Configurar el módulo de cámara del dispositivo

En este paso, se instala y habilita el módulo de cámara para su dispositivo. Ejecute el siguiente comando en el dispositivo.

Raspberry Pi (Armv7l)

1. Instale la interfaz `picamera` del módulo de cámara. Ejecute el siguiente comando para instalar el módulo de cámara y las demás bibliotecas de Python que sean necesarias para este tutorial.

   ```
   sudo apt-get install -y python3-picamera
   ```

2. Compruebe que Picamera se haya instalado correctamente.

   ```
   sudo -u ggc_user bash -c 'python3 -c "import picamera"'
   ```

   Si el resultado no contiene errores, la validación es correcta.

   > **Note**
   >
   > Si el ejecutable de Python instalado en el dispositivo es `python3.7`, utilice `python3.7` en lugar de `python3` con los comandos de este tutorial. Asegúrese de que la instalación de pip corresponde a la versión `python3.7` o `python3` correcta para evitar errores de dependencia.

3. Reinicie el dispositivo.

   ```
   sudo reboot
   ```

4. Abra la herramienta de configuración de Raspberry Pi.

   ```
   sudo raspi-config
   ```

5. Utilice las teclas de flecha para abrir Interfacing Options (Opciones de interfaz) y habilitar la interfaz de la cámara. Si se le solicita, permita que el dispositivo se reinicie.

6. Ejecute el siguiente comando para probar la configuración de la cámara.

   ```
   raspistill -v -o test.jpg
   ```

   Se abre una ventana de vista previa en el Raspberry Pi, se guarda una imagen denominada `test.jpg` en el directorio actual y se muestra información sobre la cámara en el terminal de Raspberry Pi.

7. Ejecute el siguiente comando para crear un enlace simbólico que permita que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución.

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

   El valor predeterminado de *MLRootPath* este tutorial es. */greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml La carpeta greengrass_ml_tflite_venv de esta ubicación se crea cuando implementa el componente de inferencia por primera vez en Tutorial: Realice una inferencia de clasificación de imágenes de muestra con Lite TensorFlow.

Jetson Nano (Armv8)

1. Ejecute el siguiente comando para probar la configuración de la cámara.

```
gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),
 width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink
 location=test.jpg
```

   Esto captura y guarda una imagen con el nombre test.jpg en su directorio actual.

2. (Opcional) Reinicie el dispositivo. Si tiene problemas para ejecutar el comando gst-launch del paso anterior, es posible que deba reiniciar el dispositivo para resolverlos.

```
sudo reboot
```

> **Note**
>
> En el caso de los dispositivos Armv8 (AArch64), como un Jetson Nano, no es necesario crear un enlace simbólico para permitir que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución.

Paso 2: Comprobar la suscripción al tema de notificaciones predeterminado

EnTutorial: Realice una inferencia de clasificación de imágenes de muestra con Lite TensorFlow, configuró el cliente AWS IoT MQTT que está configurado en la AWS IoT consola para ver los mensajes MQTT publicados por el componente de clasificación de imágenes de TensorFlow Lite sobre el tema. `ml/tflite/image-classification` En la AWS IoT consola, compruebe que existe esta suscripción. Si no es así, siga los pasos descritos en Paso 1: Suscribirse al tema de notificaciones predeterminado para suscribirse a este tema antes de implementar el componente en su dispositivo principal de Greengrass.

Paso 3: Modifique la configuración del componente de clasificación de imágenes de TensorFlow Lite e impleméntelo

En este paso, debe configurar e implementar el componente de clasificación de imágenes de TensorFlow Lite en su dispositivo principal:

Para configurar e implementar el componente de clasificación de imágenes de TensorFlow Lite (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, en la pestaña Componentes públicos, elija `aws.greengrass.TensorFlowLiteImageClassification`.
3. En la página aws.greengrass.TensorFlowLiteImageClassification, elija Implementar.
4. En Agregar a la implementación, elija una de las siguientes opciones:
    a. Para combinar este componente con una implementación existente en el dispositivo de destino, elija Agregar a la implementación existente y, a continuación, seleccione la implementación que desee revisar.
    b. Para crear una nueva implementación en el dispositivo de destino, elija Crear nueva implementación. Si tiene una implementación existente en su dispositivo, al elegir este paso se reemplaza la implementación existente.
5. En la página Especificar detalles, haga lo siguiente:
    a. En Información de implementación, introduzca o modifique el nombre descriptivo de su implementación.

b. En Objetivos de implementación, seleccione un objetivo para su implementación y elija Siguiente. No puede cambiar el objetivo de implementación si está revisando una implementación existente.

6. En la página Seleccionar componentes, en Componentes públicos, compruebe que el componente `aws.greengrass.TensorFlowLiteImageClassification` esté seleccionado y elija Siguiente.
7. En la página Configurar componentes, haga lo siguiente:

   a. Seleccione el componente de inferencia y elija Configurar componente.

   b. En Actualización de la configuración, introduzca la siguiente actualización de configuración en el cuadro Configuración para combinar.

   ```
   {
     "InferenceInterval": "60",
     "UseCamera": "true"
   }
   ```

   Con esta actualización de configuración, el componente accede al módulo de cámara del dispositivo y realiza inferencias a partir de las imágenes tomadas por la cámara. El código de inferencia se ejecuta cada 60 segundos.

   c. Seleccione Confirmar y, a continuación, elija Siguiente.
8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Revisar, elija Implementar.

Para configurar e implementar el componente de clasificación de imágenes de TensorFlow Lite (AWS CLI)

1. Cree un `deployment.json` archivo para definir la configuración de despliegue del componente de clasificación de imágenes de TensorFlow Lite. Este archivo debería tener el siguiente aspecto:

   ```
   {
     "targetArn":"targetArn",
     "components": {
       "aws.greengrass.TensorFlowLiteImageClassification": {
         "componentVersion": 2.1.0,
   ```

```
      "configurationUpdate": {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
    }
  }
}
```

- En el campo `targetArn`, sustituya *`targetArn`* por el Nombre de recurso de Amazon (ARN) del objeto o grupo de objetos a la que apunte la implementación, en el siguiente formato:
  - Cosa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
  - Grupo de cosas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
- Este tutorial utiliza la versión 2.1.0 de componente. En el objeto `aws.greengrass.TensorFlowLiteImageClassification` componente, *`2.1.0`* sustitúyalo por una versión diferente del componente de clasificación de imágenes de TensorFlow Lite.

Con esta actualización de configuración, el componente accede al módulo de cámara del dispositivo y realiza inferencias a partir de las imágenes tomadas por la cámara. El código de inferencia se ejecuta cada 60 segundos. Reemplace los siguientes valores

2. Ejecute el siguiente comando para implementar el componente de clasificación de imágenes TensorFlow Lite en el dispositivo:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

La implementación puede tardar varios minutos en completarse. En el siguiente paso, compruebe el registro de componentes para comprobar que la implementación se ha completado correctamente y para ver los resultados de la inferencia.

Paso 4: Visualizar los resultados de la inferencia

Tras implementar el componente, puede ver los resultados de la inferencia en el registro del componente de su dispositivo principal de Greengrass y en AWS IoT el cliente MQTT de la consola. AWS IoT Para suscribirse al tema sobre el que el componente publica los resultados de las inferencias, consulte Paso 2: Comprobar la suscripción al tema de notificaciones predeterminado.

- AWS IoT Cliente MQTT: para ver los resultados que el componente de inferencia publica en el tema de notificaciones predeterminado, complete los siguientes pasos:

  1. En el menú de navegación de la consola de AWS IoT, seleccione Prueba, cliente de prueba de MQTT.
  2. En Suscripciones, elija **`ml/tflite/image-classification`**.
- Registro de componentes: para ver los resultados de la inferencia en el registro de componentes, ejecute el siguiente comando en el dispositivo principal de Greengrass.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Si no puede ver los resultados de la inferencia en el registro de componentes o en el cliente MQTT, significa que la implementación falló o no llegó al dispositivo principal. Esto puede ocurrir si el dispositivo principal no está conectado a Internet o no tiene los permisos necesarios para ejecutar el componente. Ejecute el siguiente comando en su dispositivo principal para ver el archivo de registro del software AWS IoT Greengrass principal. Este archivo incluye registros del servicio de implementación del dispositivo principal de Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Para obtener más información, consulte Resolución de problemas de inferencia de machine learning.

Siguientes pasos

En este tutorial, se muestra cómo utilizar el componente de clasificación de imágenes TensorFlow Lite, con opciones de configuración personalizadas para realizar una clasificación de imágenes de muestra en imágenes tomadas por una cámara.

Para obtener más información sobre la personalización de la configuración de los componentes públicos o la creación de componentes de machine learning personalizados, consulte Personalización de sus componentes de machine learning.

Componentes

Los componentes de AWS IoT Greengrass son módulos de software que se implementan en los dispositivos principales de Greengrass. Los componentes pueden ser aplicaciones, instaladores en tiempo de ejecución, bibliotecas o cualquier código que se ejecute en un dispositivo. Puede definir componentes que dependan de otros componentes. Por ejemplo, puede definir un componente que instale Python y, luego, definir ese componente como una dependencia de los componentes que ejecutan aplicaciones de Python. Cuando implementa los componentes en las flotas de dispositivos, Greengrass implementa solo los módulos de software que requieren sus dispositivos.

Temas

- Componentes proporcionados por AWS
- Componentes compatibles con Publisher
- Componentes de la comunidad
- Herramientas de desarrollo de AWS IoT Greengrass
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos

Componentes proporcionados por AWS

AWS IoT Greengrass proporciona y mantiene componentes prediseñados que puede implementa en sus dispositivos. Estos componentes incluyen características (como el administrador de flujos), conectores de AWS IoT Greengrass V1 (como las métricas de CloudWatch) y herramientas de desarrollo local (como la CLI de AWS IoT Greengrass). Puede implementar estos componentes en sus dispositivos para que funcionen de forma independiente, o puede usarlos como dependencias en sus componentes personalizados de Greengrass.

> **Note**
>
> Varios de los componentes proporcionados por AWS dependen de versiones secundarias específicas del núcleo de Greengrass. Debido a esta dependencia, es necesario actualizar estos componentes al actualizar el núcleo de Greengrass a una nueva versión secundaria. Para obtener información sobre las versiones específicas del núcleo de las que depende cada componente, consulte el tema del componente correspondiente. Para más información

sobre la actualización del núcleo, consulte Actualización del software AWS IoT Greengrass Core (OTA).

Cuando un componente tiene un tipo de componente genérico y de Lambda, la versión actual del componente es del tipo genérico y la versión anterior del componente es del tipo de Lambda.

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Núcleo de Greengrass	El núcleo del software AWS IoT Greengrass Core. Utilice este componente para configurar y actualizar el software de sus dispositivos principales.	Núcleo	Linux, Windows	Sí	No
Versión lite del núcleo de Greengrass	Un núcleo ligero para dispositivos con recursos limitados, optimizado para dispositivos de periferia	NucleusLite	Linux	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
	de bajo costo y aplicaciones de gran volumen				
Autenticación del dispositivo de cliente	Habilita los dispositivos IoT locales, denominados dispositivos de cliente, a conectarse al dispositivo principal.	Complemento	Linux, Windows	Sí	No
CloudWatch métricas	Publica métricas personalizadas en Amazon CloudWatch.	Genérico, Lambda	Linux, Windows	Sí	Sí

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
AWS IoT Device Defender	Notifica a los administradores de los cambios en el estado de un dispositivo principal de Greengrass para identificar comportamientos inusuales.	Genérico, Lambda	Linux, Windows	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Spooler de disco	Habilita una opción de almacenamiento persistente para los mensajes enviados desde los dispositivos principales de Greengrass a AWS IoT Core. Este componente almacenará estos mensajes salientes en el disco.	Complemento	Linux, Windows	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Administrador de aplicaciones de Docker	Permite a AWS IoT Greengrass descargar imágenes de Docker desde Docker Hub y Amazon Elastic Container Registry (Amazon ECR).	Genérico	Linux, Windows	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Conector periférico para Kinesis Video Streams	Lee las transmisiones de video de las cámaras locales, publica los flujos en Kinesis Video Streams y los muestra en los paneles de Grafana con TwinMaker AWS IoT.	Genérico	Linux	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
CLI de Greengrass	Proporciona una interfaz de línea de comandos que puede usar para crear implementaciones locales e interactuar con el dispositivo principal de Greengrass y sus componentes.	Complemento	Linux, Windows	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Detector de IP	Envía la información de conectividad del agente MQTT a AWS IoT Greengrass, para que los dispositivos de cliente puedan descubrir cómo conectarse.	Complemento	Linux, Windows	Sí	No
Firehose	Publica datos a través de los flujos de entrega de Amazon Data Firehose a los destinos de la Nube de AWS.	Lambda	Linux	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Lanzador de Lambda	Maneja los procesos y la configuración del entorno para las funciones de Lambda.	Genérico	Linux	No	No
Administrador de Lambda	Maneja la comunicación y el escalado entre procesos para las funciones de Lambda.	Complemento	Linux	No	No
Tiempos de ejecución de Lambda	Ofrece artefactos para cada tiempo de ejecución de Lambda.	Genérico	Linux	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Enrutador de suscripción antigua	Administra las suscripciones para las funciones de Lambda que se ejecutan en AWS IoT Greengrass V1.	Genérico	Linux	No	No
Consola de depuración local	Proporciona una consola local que puede usar para depurar y administrar el dispositivo principal de Greengrass y sus componentes.	Complemento	Linux, Windows	Sí	No
Administrador de registros	Recopila y carga los registros en el dispositivo principal de Greengrass.	Complemento	Linux, Windows	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Componentes de machine learning	Proporciona modelos de machine learning y código de inferencia de muestra que puede usar para realizar inferencias de machine learning en los dispositivos principales de Greengrass.	Consulte Componentes de machine learning.			No
Adaptador de protocolo Modbus-RTU	Extrae información de dispositivos Modbus RTU locales.	Lambda	Linux	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Emisor de telemetría del núcleo	Publica los datos de telemetría del estado del sistema recopilados desde el núcleo hasta un tema local o un tema de MQTT AWS IoT Core.	Complemento	Linux, Windows	Sí	No
Puente MQTT	Retransmite mensajes MQTT entre los dispositivos cliente, los de publicación/suscripción local de AWS IoT Greengrass e AWS IoT Core.	Complemento	Linux, Windows	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Agente MQTT 3.1.1 (Moquette)	Ejecuta un agente MQTT 3.1.1 que administra los mensajes entre los dispositivos de cliente y el dispositivo principal.	Complemento	Linux, Windows	Sí	No
Agente MQTT 5 (EMQX)	Ejecuta un agente MQTT 5 que administra los mensajes entre los dispositivos de cliente y el dispositivo principal.	Genérico	Linux, Windows	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Proveedor PKCS#11	Permite que los componentes de Greengrass accedan a una clave privada y un certificado que se almacenan de forma segura en un módulo de seguridad de hardware (HSM).	Complemento	Linux	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Administrador de secretos	Implementa secretos a partir de secretos de AWS Secrets Manager para que pueda utilizar de forma segura las credenciales, como las contraseñas, en componentes personalizados del dispositivo principal de Greengrass.	Complemento	Linux, Windows	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Tunelización segura	Permite conexiones de tunelización AWS IoT seguras que puede utilizar para establecer comunicaciones bidireccionales con los dispositivos principales de Greengrass que se encuentran detrás de firewalls restringidos.	Genérico	Linux	No	Sí

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Administrador de sombras	Permite la interacción con las sombras del dispositivo principal. Administra el almacenamiento de documentos de sombra y también la sincronización de los estados de sombras locales con el servicio de sombra de dispositivo de AWS IoT.	Complemento	Linux, Windows	Sí	No
Amazon SNS	Publica mensajes en un tema de Amazon SNS.	Lambda	Linux	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Administrador de flujos	Transmite grandes volúmenes de datos de orígenes locales a la Nube de AWS.	Genérico	Linux, Windows	No	Sí
Reenviador de registros del sistema	Cargue los registros de systemd-journald a la Nube de AWS.	Genérico	Linux	Sí	Sí
Agente de Systems Manager	Administre los dispositivos principales con AWS Systems Manager, lo que le permite parchear dispositivos, ejecutar comandos y más.	Genérico	Linux	Sí	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
Servicio de intercambio de token	Proporciona credenciales de AWS que puede usar para interactuar con los servicios de AWS.	Genérico	Linux, Windows	No	No
Colector IoT SiteWise OPC UA	Recopila datos de los servidores OPC-UA.	Genérico	Linux, Windows	No	No
Simulador de fuente de datos IoT SiteWise OPC UA	Ejecuta un servidor OPC-UA local que genera datos de muestra.	Genérico	Linux, Windows	No	No
SiteWise Publicador de IoT	Publica datos en la nube de AWS.	Genérico	Linux, Windows	No	No

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto	Compatible con la versión lite del núcleo
SiteWise Procesador IoT	Procesa los datos de los dispositivos principales de Greengrass.	Genérico	Linux, Windows	No	No

Núcleo de Greengrass

El componente núcleo de Greengrass (`aws.greengrass.Nucleus`) es un componente obligatorio y el requisito mínimo para ejecutar el software AWS IoT Greengrass Core en un dispositivo. Puede configurar este componente para personalizar y actualizar el software AWS IoT Greengrass Core de forma remota. Implemente este componente para configurar ajustes como el proxy, la función del dispositivo y la configuración de las AWS IoT cosas en sus dispositivos principales.

> **Note**
>
> Desde la versión 2.14.0 de Greengrass, estará disponible una versión optimizada del tiempo de ejecución del dispositivo del núcleo para el uso de memoria de los dispositivos de periferia restringidos. Consulte la versión lite del núcleo de Greengrass para obtener más información sobre su configuración y uso.

> **Important**
>
> Cuando la versión del componente núcleo cambia, o cuando cambias determinados parámetros de configuración, el software AWS IoT Greengrass Core, que incluye el núcleo y todos los demás componentes del dispositivo, se reinicia para aplicar los cambios.
> Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se

implementen automáticamente en sus dispositivos principales si agrega nuevos dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

Temas

- Versiones
- Requisitos de los dispositivos
- Plataformas admitidas
- Sistema operativo
- Requisitos
- Dependencias
- Descarga e instalación
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.16.x
- 2.15.x
- 2.14.x
- 2.13.x
- 2.12.x
- 2.11.x

- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Requisitos de los dispositivos

> **Note**
>
> Puede utilizar AWS IoT Device Tester for AWS IoT Greengrass para comprobar que su dispositivo puede ejecutar el software AWS IoT Greengrass principal y comunicarse con el. Nube de AWS Para obtener más información, consulte Uso de AWS IoT Device Tester para la versión 2 de AWS IoT Greengrass.

Linux

- El uso de un Región de AWSque admita AWS IoT Greengrass V2. Para ver una lista completa de las regiones compatibles, consulte AWS IoT Greengrass V2 endpoints and quotas en la Referencia general de AWS.
- Un mínimo de 256 MB de espacio en disco disponible para el software AWS IoT Greengrass Core. Este requisito no incluye los componentes implementados en el dispositivo principal.
- Se asigna un mínimo de 96 MB de RAM al software AWS IoT Greengrass principal. Este requisito no incluye los componentes implementados en el dispositivo principal. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.
- Entorno de ejecución de Java (JRE) versión 8 o posterior. Java debe estar disponible en la variable de entorno PATH en el dispositivo. Para utilizar Java para desarrollar componentes

personalizados, debe instalar un kit de desarrollo de Java (JDK). Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.

- Biblioteca C de GNU (glibc) versión 2.25 o posterior.
- Debe ejecutar el software AWS IoT Greengrass Core como usuario root. Por ejemplo, utilice `sudo`.
- El usuario root que ejecuta el software AWS IoT Greengrass principal, por ejemplo`root`, debe tener permiso para ejecutar `sudo` con cualquier usuario y grupo. El archivo `/etc/sudoers` debe conceder permiso a este usuario para que se ejecute `sudo` como otros grupos. El permiso para el usuario en `/etc/sudoers` debería verse como el siguiente ejemplo.

  ```
  root    ALL=(ALL:ALL) ALL
  ```

- El dispositivo principal debe poder realizar solicitudes salientes a un conjunto de puntos de conexión y puertos. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.
- El directorio `/tmp` debe montarse con permisos `exec`.
- Todos los siguientes intérprete de comandos:
  - `ps -ax -o pid,ppid`
  - `sudo`
  - `sh`
  - `kill`
  - `cp`
  - `chmod`
  - `rm`
  - `ln`
  - `echo`
  - `exit`
  - `id`
  - `uname`
  - `grep`
- Es posible que su dispositivo también requiera los siguientes intérprete de comandos opcionales:

  - (Opcional)`systemctl`. Este comando se utiliza para configurar el software AWS IoT Greengrass Core como un servicio del sistema.
  - (Opcional) `useradd`, `groupadd` y `usermod`. Estos comandos se utilizan para configurar el usuario `ggc_user` y el grupo `ggc_group` del sistema.
  - (Opcional)`mkfifo`. Este comando se utiliza para ejecutar funciones de Lambda como componentes.
- Para configurar los límites de recursos del sistema para los procesos de los componentes, el dispositivo debe ejecutar la versión 2.6.24 o posterior del kernel de Linux.
- Para ejecutar las funciones de Lambda, el dispositivo debe cumplir requisitos adicionales. Para obtener más información, consulte Requisitos de la función de Lambda.

Windows

- El uso de una Región de AWSque soporte AWS IoT Greengrass V2. Para ver una lista completa de las regiones compatibles, consulte AWS IoT Greengrass V2 endpoints and quotas en la Referencia general de AWS.
- Un mínimo de 256 MB de espacio en disco disponible para el software AWS IoT Greengrass Core. Este requisito no incluye los componentes implementados en el dispositivo principal.
- Se asigna un mínimo de 160 MB de RAM al software AWS IoT Greengrass principal. Este requisito no incluye los componentes implementados en el dispositivo principal. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.
- Entorno de ejecución de Java (JRE) versión 8 o posterior. Java debe estar disponible en la variable de entorno PATH en el dispositivo. Para utilizar Java para desarrollar componentes personalizados, debe instalar un kit de desarrollo de Java (JDK). Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.

  > **Note**
  >
  > Para utilizar la versión 2.5.0 del núcleo de Greengrass, debe utilizar una versión de 64 bits del Entorno de ejecución de Java (JRE). La versión 2.5.1 del núcleo de Greengrass admite 32 y 64 bits. JREs

- El usuario que instala el software AWS IoT Greengrass Core debe ser administrador.

- Debe instalar el software AWS IoT Greengrass Core como un servicio del sistema. Especifique `--setup-system-service true` cuándo va a instalar el software.
- Cada usuario que ejecute los procesos componentes debe existir en la LocalSystem cuenta y el nombre y la contraseña del usuario deben estar en la instancia de Credential Manager de la LocalSystem cuenta. Puede configurar este usuario siguiendo las instrucciones para instalar el software AWS IoT Greengrass principal.
- El dispositivo principal debe poder realizar solicitudes salientes a un conjunto de puntos de conexión y puertos. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

Plataformas admitidas

AWS IoT Greengrass oficialmente es compatible con dispositivos que ejecutan las siguientes plataformas. Es posible que los dispositivos con plataformas no incluidas en esta lista funcionen, pero AWS IoT Greengrass las pruebas solo se realizan en estas plataformas especificadas.

Linux

Arquitecturas:

- Armv7l
- Armv8 () AArch64
- x86_64

Windows

Arquitecturas:

- x86_64

Versiones:

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

> **Note**
>
> Algunas AWS IoT Greengrass funciones no son compatibles actualmente con los dispositivos Windows. Para obtener más información, consulte Compatibilidad de características de Greengrass y Aspectos a tener en cuenta sobre las características.

Aspectos a tener en cuenta sobre las características

Algunas AWS IoT Greengrass funciones no son compatibles actualmente con los dispositivos Windows. Revise las diferencias de las características para confirmar si un dispositivo Windows cumple sus requisitos. Para obtener más información, consulte Compatibilidad de características de Greengrass.

Para crear un sistema operativo personalizado basado en Linux, puedes usar la BitBake receta AWS IoT Greengrass del proyecto. `meta-aws` El `meta-aws` proyecto proporciona recetas que puede utilizar para desarrollar capacidades de software de AWS vanguardia en sistemas Linux integrados que se crean con OpenEmbeddedlos marcos de compilación del Proyecto Yocto. Yocto Project es un proyecto de colaboración de código abierto que le ayuda a crear sistemas personalizados basados en Linux para aplicaciones incrustadas independientemente de la arquitectura del hardware. La BitBake receta para AWS IoT Greengrass instalar, configurar y ejecutar automáticamente el software AWS IoT Greengrass Core en su dispositivo.

Las plataformas Linux también se pueden ejecutar AWS IoT Greengrass en un contenedor Docker. Para obtener más información, consulte Ejecute AWS IoT Greengrass el software principal en un contenedor de Docker.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Para obtener más información, consulte Plataformas admitidas.

Requisitos

Los dispositivos deben cumplir ciertos requisitos para instalar y ejecutar el núcleo de Greengrass y el software AWS IoT Greengrass Core. Para obtener más información, consulte Requisitos de los dispositivos.

Se admite la ejecución del componente núcleo de Greengrass en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.

- El componente núcleo de Greengrass debe tener conectividad con AWS IoT data, AWS IoT Credentials y Amazon S3.

Dependencias

El núcleo de Greengrass no incluye ninguna dependencia de componentes. Sin embargo, varios componentes proporcionados por AWS incluyen el núcleo como dependencia. Para obtener más información, consulte Componentes proporcionados por AWS.

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Descarga e instalación

Puede descargar un instalador que configura el componente núcleo de Greengrass en el dispositivo. Este instalador configura el dispositivo como un dispositivo principal de Greengrass. Hay dos tipos de instalaciones que puede realizar: una instalación rápida, que crea AWS los recursos necesarios, o una instalación manual, en la que puede crear los AWS recursos usted mismo. Para obtener más información, consulte Instalación del software AWS IoT Greengrass Core.

También puede seguir un tutorial para instalar el núcleo de Greengrass y explorar el desarrollo de componentes de Greengrass. Para obtener más información, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente. Algunos parámetros requieren que el software AWS IoT Greengrass principal se reinicie para que surta efecto. Para obtener más información acerca de por qué y cómo configurar este componente, consulte Configurar el software AWS IoT Greengrass principal.

`iotRoleAlias`

El alias del AWS IoT rol que apunta a un rol de IAM de intercambio de fichas. El proveedor de AWS IoT credenciales asume esta función para permitir que el dispositivo principal de Greengrass interactúe con AWS los servicios. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Al ejecutar el software AWS IoT Greengrass Core con la `--provision true` opción, el software proporciona un alias de rol y establece su valor en el componente core.

`interpolateComponentConfiguration`

(Opcional) Puede habilitar el núcleo de Greengrass para interpolar las variables de las recetas de los componentes en las configuraciones y combinar las actualizaciones de configuración. Se recomienda configurar esta opción en `true` para que el dispositivo principal pueda ejecutar componentes de Greengrass que usen variables de receta en sus configuraciones.

Esta característica está disponible en la versión 2.6.0 y posteriores de este componente.

Valor predeterminado: `false`

`networkProxy`

(Opcional) El proxy de red que se utilizará en todas las conexiones. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

> **Important**
>
> Al implementar un cambio en este parámetro de configuración, el software AWS IoT Greengrass principal se reinicia para que el cambio surta efecto.

Este objeto contiene la siguiente información:

`noProxyAddresses`

(Opcional) Una lista separada por comas de direcciones IP o nombres de host que están exentos del proxy.

`proxy`

El proxy al que conectar. Este objeto contiene la siguiente información:

`url`

La dirección URL del servidor proxy, en el formato `scheme://userinfo@host:port`.

- `scheme`: el esquema, que debe ser `http` o `https`.

  > **Important**
  >
  > Los dispositivos principales de Greengrass deben ejecutar la versión 2.5.0 o versiones posteriores del núcleo de Greengrass para usar proxies HTTPS. Si configura un proxy HTTPS, debe agregar el certificado de la CA del servidor proxy al certificado de la CA raíz de Amazon del dispositivo principal. Para obtener más información, consulte Permita que el dispositivo principal confíe en un proxy HTTPS.

- `userinfo`: (opcional) la información de nombre de usuario y contraseña. Si especifica esta información en `url`, el dispositivo principal de Greengrass ignora los campos `username` y `password`.
- `host`: el nombre de host o dirección IP del servidor proxy.
- `port`: (opcional) el número de puerto. Si no especifica el puerto, el dispositivo principal de Greengrass usará los siguientes valores predeterminados:
  - `http`: 80
  - `https`: 443

`username`

(Opcional) El nombre de usuario que autentica el servidor proxy.

`password`

(Opcional) La contraseña para autenticarse en el servidor proxy.

`mqtt`

(Opcional) La configuración MQTT del dispositivo principal de Greengrass. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

> **Important**
>
> Al implementar un cambio en este parámetro de configuración, el software AWS IoT Greengrass principal se reinicia para que el cambio surta efecto.

Este objeto contiene la siguiente información:

`port`

(Opcional) El puerto que se utilizará para las conexiones MQTT.

Valor predeterminado: `8883`

`keepAliveTimeoutMs`

(Opcional) El tiempo en milisegundos que transcurre entre cada mensaje `PING` que envía el cliente para mantener activa la conexión MQTT. El valor debe ser mayor que `pingTimeoutMs`.

Valor predeterminado: `60000` (60 segundos).

`pingTimeoutMs`

(Opcional) El tiempo en milisegundos que el cliente espera para recibir un mensaje `PINGACK` del servidor. Si la espera supera el tiempo de espera, el dispositivo principal se cierra y vuelve a abrir la conexión MQTT. Este valor debe ser inferior a `keepAliveTimeoutMs`.

Valor predeterminado: `30000` (30 segundos)

`operationTimeoutMs`

(Opcional) El tiempo en milisegundos que el cliente espera a que finalicen las operaciones de MQTT (por ejemplo, `CONNECT` o `PUBLISH`). Esta opción no se aplica al `PING` MQTT ni a los mensajes de keep alive.

Valor predeterminado: `30000` (30 segundos).

`maxInFlightPublishes`

(Opcional) La cantidad máxima de mensajes de QoS 1 de MQTT sin confirmar que pueden estar en proceso al mismo tiempo.

Esta característica está disponible en la versión 2.1.0 y posteriores de este componente.

Valor predeterminado: `5`

Rango válido: valor máximo de 100

`maxMessageSizeInBytes`

(Opcional) El tamaño máximo de un mensaje MQTT. Si un mensaje supera este tamaño, el núcleo de Greengrass lo rechaza con error.

Esta característica está disponible en la versión 2.1.0 y posteriores de este componente.

Valor predeterminado: `131072` (128 KB)

Rango válido: valor máximo de `2621440` (2,5 MB)

`maxPublishRetry`

(Opcional) El número máximo de reintentos permitidos para un mensaje que aun no se publicó. Puede especificar `-1` para volver a intentar un número ilimitado de veces.

Esta característica está disponible en la versión 2.1.0 y posteriores de este componente.

Valor predeterminado: `100`

`spooler`

(Opcional) La configuración del spooler MQTT del dispositivo principal de Greengrass. Este objeto contiene la siguiente información:

`storageType`

El tipo de almacenamiento para almacenar los mensajes. Si `storageType` está establecido en `Disk`, se puede configurar el `pluginName`. Puede especificar `Memory` o `Disk`.

Esta característica está disponible para la versión 2.11.0 y versiones posteriores del componente núcleo de Greengrass.

> **Important**
>
> Si el spooler de MQTT `storageType` está configurado en `Disk` y desea revertir el núcleo de Greengrass de la versión 2.11.x a una versión anterior, debe volver a cambiar la configuración a `Memory`. La única configuración para `storageType` compatible con las versiones 2.10.x y anteriores del núcleo de Greengrass es `Memory`. Si no se siguen estas instrucciones, se puede romper el spooler. Esto provocaría que el dispositivo principal de Greengrass no pudiera enviar mensajes MQTT a la Nube de AWS.

Valor predeterminado: `Memory`

`pluginName`

(Opcional) El nombre del componente del complemento. Este componente solo se usará si `storageType` está configurado en `Disk`. Esta opción se predetermina en `aws.greengrass.DiskSpooler` y usará el Spooler de disco proporcionado por Greengrass.

Esta característica está disponible para la versión 2.11.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `"aws.greengrass.DiskSpooler"`

`maxSizeInBytes`

(Opcional) El tamaño máximo de la memoria caché en la que el dispositivo principal almacena los mensajes MQTT sin procesar. Si la caché está llena, los mensajes nuevos se rechazan.

Valor predeterminado: `2621440` (2,5 MB)

`keepQos0WhenOffline`

(Opcional) Puede agrupar los mensajes de QoS 0 de MQTT recibidos en el dispositivo principal mientras está desconectado. Si establece esta opción en `true`, el dispositivo principal almacena los mensajes de QoS 0 que no puede enviar mientras está desconectado. Si establece esta opción en `false`, el dispositivo principal descartará estos mensajes. El dispositivo principal siempre envía los mensajes de QoS 1 a menos que el spool esté lleno.

Valor predeterminado: `false`

`version`

(Opcional) La versión de MQTT. Puede especificar `mqtt3` o `mqtt5`.

Esta característica está disponible para la versión 2.10.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `mqtt5`

`receiveMaximum`

(Opcional) La cantidad máxima de paquetes QoS 1 no confirmados que puede enviar el agente.

Esta característica está disponible para la versión 2.10.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `100`

`sessionExpirySeconds`

(Opcional) La cantidad de tiempo en segundos que puede solicitar para que dure una sesión desde IoT Core. El valor predeterminado es el tiempo máximo admitido por AWS IoT Core.

Esta característica está disponible para la versión 2.10.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `604800 (7 days)`

`minimumReconnectDelaySeconds`

(Opcional) Una opción para el comportamiento de reconexión. Tiempo mínimo en segundos para que MQTT se vuelva a conectar.

Esta característica está disponible para la versión 2.10.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `1`

`maximumReconnectDelaySeconds`

(Opcional) Una opción para el comportamiento de reconexión. Tiempo máximo en segundos que MQTT se vuelve a conectar.

Esta característica está disponible para la versión 2.10.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `120`

`minimumConnectedTimeBeforeRetryResetSeconds`

(Opcional) Una opción para el comportamiento de reconexión. La cantidad de tiempo en segundos que una conexión debe estar activa antes de que el aplazo en el reintento se restablezca al mínimo.

Esta característica está disponible para la versión 2.10.0 y versiones posteriores del componente núcleo de Greengrass.

Valor predeterminado: `30`

`jvmOptions`

(Opcional) Las opciones de JVM que se utilizarán para ejecutar el software AWS IoT Greengrass principal. Para obtener información sobre las opciones de JVM recomendadas para ejecutar el software AWS IoT Greengrass Core, consulte. Control de la asignación de memoria con las opciones de JVM

> **Important**
>
> Al implementar un cambio en este parámetro de configuración, el software AWS IoT Greengrass principal se reinicia para que el cambio surta efecto.

`iotDataEndpoint`

El punto final AWS IoT de datos para su. Cuenta de AWS

Cuando ejecuta el software AWS IoT Greengrass Core con la `--provision true` opción, el software obtiene sus datos y credenciales de los puntos finales AWS IoT y los coloca en el componente núcleo.

`iotCredEndpoint`

El punto final de AWS IoT credenciales para su. Cuenta de AWS

Cuando ejecuta el software AWS IoT Greengrass Core con la `--provision true` opción, el software obtiene sus datos y credenciales de los puntos finales AWS IoT y los establece en el componente núcleo.

`greengrassDataPlaneEndpoint`

Esta característica está disponible en la versión 2.7.0 y posteriores de este componente.

Para obtener más información, consulte Use un certificado de dispositivo firmado por una CA privada.

`greengrassDataPlanePort`

Esta característica está disponible en la versión 2.0.4 y posteriores de este componente.

(Opcional) El puerto que se utilizará para las conexiones del plano de datos. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

> **Important**
>
> Debe especificar un puerto en el que el dispositivo pueda realizar conexiones salientes. Si especificas un puerto que está bloqueado, el dispositivo no podrá conectarse AWS IoT Greengrass para recibir las implementaciones.

Puede elegir entre las siguientes opciones:

- `443`
- `8443`

Valor predeterminado: `8443`

`awsRegion`

El que se Región de AWS debe usar.

`runWithDefault`

El usuario del sistema que se utilizará para ejecutar los componentes.

> **Important**
>
> Al implementar un cambio en este parámetro de configuración, el software AWS IoT Greengrass principal se reinicia para que el cambio surta efecto.

Este objeto contiene la siguiente información:

`posixUser`

El nombre o ID del usuario del sistema y, opcionalmente, del grupo de sistemas que el dispositivo principal utiliza para ejecutar los componentes genéricos y de Lambda. Especifique el usuario y el grupo separados por dos puntos (`:`) con el siguiente formato: `user:group`. El grupo es opcional. Si no especifica un grupo, el software AWS IoT Greengrass Core utiliza el grupo principal para el usuario. Por ejemplo, puede especificar `ggc_user` o `ggc_user:ggc_group`. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

Al ejecutar el instalador del software AWS IoT Greengrass Core con la `--component-default-user` *`ggc_user`*`:`*`ggc_group`* opción, el software establece este parámetro en el componente core.

`windowsUser`

Esta característica está disponible en la versión 2.5.0 y posteriores de este componente.

El nombre del usuario de Windows que se utilizará para ejecutar este componente en los dispositivos principales de Windows. El usuario debe estar en todos los dispositivos principales de Windows y su nombre y contraseña deben almacenarse en la instancia del administrador de credenciales de la LocalSystem cuenta. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

Al ejecutar el instalador de software AWS IoT Greengrass Core con la `--component-default-user` *`ggc_user`* opción, el software establece este parámetro en el componente core.

`systemResourceLimits`

Esta característica solo está disponible en la versión 2.4.0 o posterior de este componente. AWS IoT Greengrass actualmente no admite esta característica en los dispositivos principales de Windows.

Los límites de recursos del sistema se aplicarán de forma predeterminada a los procesos de componentes de Lambda genéricos y no contenerizados. Puede anular los límites de recursos del sistema para los componentes individuales cuando crea una implementación. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes.

Este objeto contiene la siguiente información:

`cpus`

La cantidad máxima de tiempo de CPU que los procesos de cada componente pueden utilizar en el dispositivo principal. El tiempo total de CPU de un dispositivo principal equivale a la cantidad de núcleos de CPU del dispositivo. Por ejemplo, en un dispositivo principal con 4 núcleos de CPU, puede establecer este valor en 2 para limitar los procesos de cada componente al 50 % de uso de cada núcleo de CPU. En un dispositivo con 1 núcleo de CPU, puede establecer este valor en `0.25` para limitar los procesos de cada componente al 25 % de uso de la CPU. Si establece este valor en un número superior al

número de núcleos de la CPU, el software AWS IoT Greengrass Core no limita el uso de la CPU de los componentes.

`memory`

La cantidad máxima de RAM, expresada en kilobytes, que los procesos de un componente pueden utilizar en el dispositivo principal.

`s3EndpointType`

(Opcional) El tipo de punto de conexión de S3. Este parámetro solo tendrá efecto en la región Este de EE. UU. (Norte de Virginia) (`us-east-1`). Se ignorará la configuración de este parámetro desde cualquier otra región. Puede elegir entre las siguientes opciones:

- `REGIONAL`: el cliente de S3 y la URL prefirmada utilizan el punto de conexión regional.
- `GLOBAL`: el cliente de S3 y la URL prefirmada utilizan el punto de conexión heredado.
- `DUALSTACK`: la URL prefirmada de S3 utiliza el punto de conexión de doble pila.

Valor predeterminado: `GLOBAL`

`fipsMode`

(Opcional) Hace que Greengrass utilice puntos de conexión de FIPS. Para obtener más información sobre cómo habilitar los puntos de conexión de FIPS, consulte Puntos de conexión de FIPS.

Puede elegir entre las siguientes opciones:

- Si `true` se establece en verdadero, los puntos de conexión utilizarán el punto de conexión de FIPS.
- Si `false` se establece en falso, los puntos de conexión no utilizarán el punto de conexión de FIPS.

Valor predeterminado: `false`

`logging`

(Opcional) La configuración de registro del dispositivo principal. Para obtener más información acerca de cómo configurar y usar los registros de Greengrass, consulte Supervisión de los registros de AWS IoT Greengrass.

Este objeto contiene la siguiente información:

`level`

(Opcional) El nivel mínimo de mensajes de registro que se van a generar.

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`format`

(Opcional) El formato de datos de los registros. Puede elegir entre las siguientes opciones:

- `TEXT`: elija esta opción si desea ver los registros en forma de texto.
- `JSON`: elija esta opción si desea ver los registros con el comando logs de la CLI de Greengrass o interactuar con los registros mediante programación.

Valor predeterminado: `TEXT`

`outputType`

(Opcional) El tipo de salida para los registros. Puede elegir entre las siguientes opciones:

- `FILE`— El software AWS IoT Greengrass Core envía los registros a los archivos del directorio que especifique. `outputDirectory`
- `CONSOLE`— El software AWS IoT Greengrass Core imprime los registros en`stdout`. Seleccione esta opción para ver los registros a medida que los imprime el dispositivo principal.

Valor predeterminado: `FILE`

`fileSizeKB`

(Opcional) El tamaño máximo de cada archivo de registro (en kilobytes). Cuando un archivo de registro supera este tamaño máximo de archivo, el software AWS IoT Greengrass Core crea un nuevo archivo de registro.

Este parámetro solo se aplica cuando se especifica `FILE` para `outputType`.

Valor predeterminado: `1024`

`totalLogsSizeKB`

(Opcional) El tamaño total máximo de los archivos de registro (en kilobytes) de cada componente, incluido el núcleo de Greengrass. Los archivos de registro del núcleo de Greengrass también incluyen registros de los componentes del complemento. Cuando el tamaño total de los archivos de registro de un componente supera este tamaño máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de ese componente.

Este parámetro equivale al parámetro de límite de espacio en disco del componente del administrador de registros (`diskSpaceLimit`), que puede especificar para el núcleo (sistema) de Greengrass y para cada componente. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para el núcleo de Greengrass y cada componente.

Este parámetro solo se aplica cuando se especifica `FILE` para `outputType`.

Valor predeterminado: `10240`

`outputDirectory`

(Opcional) El directorio de salida para los archivos de registro.

Este parámetro solo se aplica cuando se especifica `FILE` para `outputType`.

Valor predeterminado: *`/greengrass/v2`*`/logs`, donde *`/greengrass/v2`* es la carpeta raíz de AWS IoT Greengrass .

`fleetstatus`

Este parámetro está disponible en la versión 2.1.0 y posteriores de este componente.

(Opcional) La configuración del estado de la flota para el dispositivo principal.

Este objeto contiene la siguiente información:

`periodicStatusPublishIntervalSeconds`

(Opcional) El tiempo (en segundos) transcurrido desde que el dispositivo principal publica el estado del dispositivo en la Nube de AWS.

Mínimo: `86400` (24 horas)

Valor predeterminado: `86400` (24 horas)

`telemetry`

(Opcional) La configuración de telemetría del estado del sistema para el dispositivo principal. Para obtener más información sobre las métricas de telemetría y cómo actuar en función de los datos de telemetría, consulte Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass.

Este objeto contiene la siguiente información:

`enabled`

(Opcional) Puede habilitar o deshabilitar la telemetría.

Valor predeterminado: `true`

`periodicAggregateMetricsIntervalSeconds`

(Opcional) El intervalo (en segundos) durante el que el dispositivo principal agrega las métricas.

Si establece este valor por debajo del valor mínimo admitido, el núcleo utilizará el valor predeterminado.

Mínimo: `3600`

Valor predeterminado: `3600`

`periodicPublishMetricsIntervalSeconds`

(Opcional) El tiempo (en segundos) transcurrido desde que el dispositivo principal publica las métricas de telemetría en la Nube de AWS.

Si establece este valor por debajo del valor mínimo admitido, el núcleo utilizará el valor predeterminado.

Mínimo: `86400`

Valor predeterminado: `86400`

`deploymentPollingFrequencySeconds`

(Opcional) El periodo en segundos durante el que se deben sondear las notificaciones de implementación.

Valor predeterminado: `15`

`componentStoreMaxSizeBytes`

(Opcional) El tamaño máximo en disco del almacén de componentes, que incluye las recetas y los artefactos de los componentes.

Valor predeterminado: `10000000000` (10 GB)

`platformOverride`

(Opcional) Un diccionario de atributos que identifica la plataforma del dispositivo principal. Utilícelo para definir los atributos de plataforma personalizados que las recetas de componentes pueden usar para identificar el ciclo de vida y los artefactos correctos del componente. Por ejemplo, puede definir un atributo de capacidad de hardware para implementar solo el conjunto mínimo de artefactos necesarios para la ejecución de un componente. Para más información, consulte el parámetro de plataforma de manifiesto en la receta del componente.

También puede usar este parámetro para anular los atributos `os` y `architecture` de la plataforma del dispositivo principal.

`httpClient`

Este parámetro está disponible en la versión 2.5.0 y posteriores de este componente.

(Opcional) La configuración del cliente HTTP del dispositivo principal. Estas opciones de configuración se aplican a todas las solicitudes HTTP realizadas por este componente. Si un dispositivo principal funciona en una red más lenta, puede aumentar los tiempos de espera para evitar que se agoten los tiempos de espera de las solicitudes HTTP.

Este objeto contiene la siguiente información:

`connectionTimeoutMs`

(Opcional) El tiempo (en milisegundos) que se debe esperar para que se abra una conexión antes de que agote el tiempo de espera de la solicitud de conexión.

Valor predeterminado: `2000` (2 segundos)

`socketTimeoutMs`

(Opcional) El tiempo (en milisegundos) que se debe esperar para que los datos se transfieran a través de una conexión abierta antes de que se agote el tiempo de espera de la conexión.

Valor predeterminado: `30000` (30 segundos)

`deploymentConfigurationTimeSource`

Este parámetro está disponible en la versión 2.15.0 y versiones posteriores de este componente.

(Opcional) La marca de tiempo que se debe usar al procesar una implementación. El valor predeterminado es `deploymentCreationTime`.

El objeto contiene los siguientes valores:

`deploymentCreationTime`

El valor predeterminado es `deploymentConfigurationTimeSource`. El dispositivo utiliza la marca de tiempo de creación de la implementación para resolver los conflictos de claves de configuración durante el procesamiento. Si se selecciona este comportamiento, la configuración del dispositivo local sostenida por el núcleo puede tener una marca temporal mayor que la de la implementación entrante, y rechaza los cambios de configuración entrantes que ahora se consideran anticuados.

`deploymentProcessingTime`

El dispositivo utiliza su marca de tiempo local para resolver los conflictos de claves de configuración durante la implementación. Cuando se procesa, el dispositivo actualiza las configuraciones según la marca de tiempo de procesamiento y no según la marca de tiempo de creación de la implementación. Este comportamiento supone que el reloj del dispositivo está calibrado correctamente.

Si desea que los nuevos dispositivos utilicen este comportamiento en la primera conexión, configure este ajuste de núcleo en la imagen o instalación inicial del dispositivo, en lugar de implementarlo mediante una implementación. Utilice la opción `--init-config` del instalador de la versión clásica del núcleo para esta configuración.

Esta configuración inicial es esencial porque los dispositivos procesan múltiples implementaciones en orden arbitrario. Si no hay una configuración inicial adecuada, es posible que un dispositivo procese las implementaciones con el comportamiento `deploymentCreationTime` predeterminado antes de recibir la implementación en la que se establece la configuración del núcleo `deploymentProcessingTime`.

Example Ejemplo: actualización de la combinación de configuraciones

```
{
```

```
  "iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "http://my-proxy-server:1100",
      "username": "Mary_Major",
      "password": "pass@word1357"
    }
  },
  "mqtt": {
    "port": 443
  },
  "greengrassDataPlanePort": 443,
  "jvmOptions": "-Xmx64m",
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la AWS IoT Greengrass carpeta raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.16.1	Mejoras y correcciones de errores • Añade una configuración para los intervalos de reintento de credencia les tras un error en el servicio Token Exchange.
2.16.0	Nuevas características • Nucleus ahora es compatible con cgroups V2. Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo no limpiaba las implement aciones obsoletas cuando se cancelaba una implementación.
2.15.1	Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo no podía limpiar los procesos de los componentes después de un tiempo de espera de 30 segundos durante las implementaciones de arranque.
2.15.0	Nuevas características • Agrega una característica de telemetría para incluir información del sistema anfitrión, como detalles de la CPU y el sistema operativo. Para obtener más información, consulte la página de componentes de telemetría. • Agrega la configuración `deploymentConfigurationTime Source` . Para obtener más información, consulte la configuración del componente del núcleo.

Versión	Cambios
	Mejoras y correcciones de errores • El núcleo ahora prioriza, por defecto, el uso de versiones y artefactos locales, lo que mejora la coherencia de la implementación y reduce las dependencias externas. Los usuarios pueden anular este comportam iento especificando preferencias alternativas en el documento de implementación. • Agrega un nuevo registro de advertencias cuando un usuario de Windows proporcionado no coincide con el conjunto de caracteres que normalmente acepta Windows.
2.14.3	Mejoras y correcciones de errores • Permite que el servicio de intercambio de tokens se reinicie al producirse cambios en la configuración del puerto. • Soluciona un problema que impedía que el servicio de estado de la flota enviara mensajes de cambio de estado de los componentes para Lambdas no fijadas y que no se activaban. • Soluciona un problema que impedía que los componentes se apagaran correctamente cuando se implementaba una nueva versión del component e. • Soluciona un problema que provocaba que los enlaces de inyección de los complementos integrados se ejecutaran dos veces, lo que hacía que se produjeran eventos adicionales durante el ciclo de vida y se duplicara el registro. • Mejora el registro del ciclo de vida de los componentes en los dispositivos Windows.
2.14.2	Mejoras y correcciones de errores • Soluciona un problema que impedía que un cliente HTTP no estuviera configurado con una autenticación mutua.

Versión	Cambios
2.14.1	Mejoras y correcciones de errores • Soluciona un problema que provocaba que los componentes no se detuvieran correctamente en las nuevas instalaciones de Greengrass.
2.14.0	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Nuevas características • El nuevo soporte para terminales de doble pila permite IPv6 la comunicación en red. • Resiliencia mejorada contra los errores de reinicio del núcleo y la corrupción de directorios `Launch`. Mejoras y correcciones de errores • Corrige las pérdidas de memoria en los cierres de PubSub suscripci ones a IPC. • Soluciona el ciclo de vida de ejecución del componente, donde entra en estado de ERROR debido al tiempo de espera de inicio cuando la condición skipif es verdadera. • Soluciona un problema por el que el dispositivo principal no se puede conectar AWS IoT Core cuando la política de TLS está configurada en _1_3_2022_10. TLS13
2.13.0	Nuevas características • Compatibilidad con puntos de conexión de FIPS en el núcleo. Mejoras y correcciones de errores • Mejoras relacionadas con la cancelación de la implementación: ahora, las implementaciones se pueden cancelar mientras se combina una nueva configuración y mientras se espera a que se inicien los servicios.

Versión	Cambios
2.12.6	Mejoras y correcciones de errores • Soluciona un problema que provocaba un bloqueo al inicio en algunos ARMv8 procesadores, incluido el Jetson Nano.
2.12.5	Mejoras y correcciones de errores • Soluciona el problema que provocaba que, en ocasiones, la reversión de la implementación se bloqueara al revertir un componente previamente dañado con dependencias sólidas. • Soluciona el problema por el que el núcleo no publicaba actualizaciones de estado tras el aprovisionamiento de la flota. • Agrega reintentos a la API de `GetDeploymentConfiguration` después de recibir errores 404.
2.12.4	Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo entra en un punto muerto durante el startup en algunos dispositivos de Linux.
2.12.3	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Soluciona un problema que provocaba que el núcleo no mostrara el estado correcto de los componentes después del relanzamiento del núcleo y durante la recuperación del componente. • Corrección de errores y mejoras generales.
2.12.2	Mejoras y correcciones de errores • Corrige un problema por el que los registros antiguos no se eliminaban correctamente. • Corrección de errores y mejoras generales.

Versión	Cambios
2.12.1	Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo podía duplicar las suscripci ones de MQTT en temas de implementación, lo que provoca más registros y publicaciones en MQTT adicionales.
2.12.0	Nuevas características • Le permite ejecutar los pasos del ciclo de vida del arranque como parte de una implementación de reversión.
2.11.3	Mejoras y correcciones de errores • Corrige un problema en el núcleo que puede iniciar incorrectamente un componente cuando fallan sus dependencias. Nuevas características • Agrega un tipo de punto de conexión de S3 configurable.
2.11.2	Mejoras y correcciones de errores • Soluciona un problema en el cliente MQTT 5 del núcleo, que podía aparecer desconectado cuando se utilizaban un gran número de suscripciones (> 50). • Agrega un reintento por el error TCP del docker dial.
2.11.1	Mejoras y correcciones de errores • Soluciona un problema por el que el núcleo no se inicia si se produce un error en una tarea de arranque y el archivo de metadatos de implement ación esta dañado. • Soluciona el problema por el que los componentes de Lambda bajo demanda no se incluyen en las actualizaciones del estado de la implementación. • Añade compatibilidad con la política de autorizaciones duplicadas. IDs

Versión	Cambios
2.11.0	Nuevas características • Le permite cancelar una implementación local. • Le permite configurar una política de gestión de errores para una implementación local. • Suma compatibilidad con un complemento del spooler de disco.
2.10.3	Mejoras y correcciones de errores • Soluciona un problema por el que Greengrass no se suscribe a las notificaciones de implementación cuando utiliza el proveedor PKCS #11.
2.10.2	Mejoras y correcciones de errores • Permite analizar los ciclos de vida de los componentes sin distinguir entre mayúsculas y minúsculas. • Soluciona el problema por el que la variable PATH del entorno no se recreaba correctamente. • Corrige la codificación URI del proxy para los componentes, incluido el administrador de flujos para los nombres de usuario con caracteres especiales.
2.10.1	Mejoras y correcciones de errores • Soluciona un problema que podía provocar un bloqueo al arrancar algunos ARMv8 procesadores, incluido el Jetson Nano. • Greengrass ya no cierra el estándar de un componente, lo que revierte el comportamiento al de la versión anterior a la 2.10.0

Versión	Cambios
2.10.0	Nuevas características • Suma compatibilidad de `interpolateComponentConfiguration` con la expresión regular vacía. Greengrass ahora interpola desde el objeto de configuración raíz. • Añade soporte para. MQTT5 • Agrega un mecanismo para cargar los componentes del complemento rápidamente sin necesidad de escanearlos. • Permite a Greengrass ahorrar espacio en disco al eliminar las imágenes de Docker no utilizadas. Mejoras y correcciones de errores • Soluciona un problema por el cual la reversión dejaba ciertos valores de configuración de una implementación en su lugar. • Soluciona un problema por el que el núcleo de Greengrass valida una secuencia de AWS dominios en puntos finales de AWS datos y sin credenciales personalizados. • Actualiza la resolución de dependencias multigrupo para volver a resolver todas las dependencias de los grupos mediante la Nube de AWS negociación, en lugar de limitarlas a la versión activa. Esta actualización también elimina el código de error de implementación `INSTALLED_COMPONENT_NOT_FOUND` . • Actualiza el núcleo de Greengrass para omitir la descarga de imágenes de Docker cuando ya existen localmente. • Actualiza el núcleo de Greengrass para reiniciar el paso de instalación de un componente antes de que se agote el tiempo de espera. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.9.6	Mejoras y correcciones de errores • Corrige un problema por el que una implementación de Greengras s fallaba con el error LAUNCH_DIRECTORY_CORRUPTED y un posterior reinicio del dispositivo no podía iniciar Greengrass. Este error puede producirse al mover el dispositivo de Greengrass entre varios grupos de elementos con implementaciones que requieren el reinicio de Greengrass.
2.9.5	Nuevas características • Suma compatibilidad con la verificación de firmas del software de núcleo de Greengrass. Mejoras y correcciones de errores • Soluciona un problema por el que una implementación falla cuando la región de metadatos de la receta local no coincide con la región de lanzamiento del núcleo de Greengrass. El núcleo de Greengrass ahora renegocia con la nube cuando esto ocurre. • Soluciona un problema por el que el spooler de mensajes de MQTT se llena y nunca elimina los mensajes. • Correcciones y mejoras menores adicionales.
2.9.4	Mejoras y correcciones de errores • Comprueba si hay un mensaje nulo antes de eliminar los mensajes de QOS 0. • Trunca los valores de detalle del estado del trabajo si superan el límite de 1024 caracteres. • Actualiza el script de arranque para Windows para que lea correctam ente la ruta raíz de Greengrass si esa ruta incluye espacios. • Actualiza la suscripción a para AWS IoT Core que se eliminen los mensajes de los clientes si no se envió la respuesta de la suscripción. • Garantiza que el núcleo cargue la configuración a partir de archivos de respaldo cuando el archivo de configuración principal esté dañado o falte.

Versión	Cambios
2.9.3	Mejoras y correcciones de errores • Garantiza que los clientes MQTT IDs no estén duplicados. • Agrega una lectura y escritura de archivos más robustas para evitar la corrupción y recuperarse de ella. • Reintenta extraer la imagen de Docker en caso de errores específicos relacionados con la red. • Agrega la opción `noProxyAddresses` de conexión MQTT.
2.9.2	Mejoras y correcciones de errores • Soluciona el problema por el que la configuración de `interpolateComponentConfiguration` no se aplicaba a una implementación en curso. • Utiliza OSHI para enumerar todos los procesos secundarios.
2.9.1	Mejoras y correcciones de errores • Agrega una corrección por la que Greengrass se reinicia si una implementación elimina un componente del complemento.
2.9.0	Nuevas características • Agrega la posibilidad de crear subimplementaciones que reintenten las implementaciones con un subconjunto de dispositivos más pequeño. Esta característica crea una forma más eficiente de probar y resolver las implementaciones fallidas. Mejoras y correcciones de errores • Mejora la compatibilidad con los sistemas que no tienen `useradd`, `groupadd` y `usermod`. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.8.1	Mejoras y correcciones de errores • Soluciona el problema por el que los códigos de error de implement ación no se generaban correctamente a partir de errores de la API de Greengrass. • Soluciona el problema que provocaba que las actualizaciones del estado de la flota enviaran información inexacta cuando un component e alcanzaba un estado `ERRORED` determinado durante una implement ación. • Soluciona el problema por el que las implementaciones no se podían completar cuando Greengrass tenía más de 50 suscripciones existente s.

Versión	Cambios
2.8.0	Nuevas características • Actualiza el núcleo de Greengrass para informar de una respuesta sobre el estado de la implementación que incluye códigos de error detallados cuando se produce un problema al implementar los componentes en un dispositivo principal. Para obtener más información, consulte Códigos de error de implementación detallados. • Actualiza el núcleo de Greengrass para informar de una respuesta al estado de un componente que incluye códigos de error detallados cuando un componente entra en el estado `BROKEN` o `ERRORED`. Para obtener más información, consulte Códigos de estado de componentes detallados. • Amplía los campos de los mensajes de estado para mejorar la informaci ón de disponibilidad de los dispositivos en la nube. • Mejora la solidez del servicio de estado de la flota. Mejoras y correcciones de errores • Permite volver a instalar un componente dañado cuando cambia su configuración. • Soluciona el problema por el que el reinicio del núcleo durante la implementación de arranque provoca un error en la implementación. • Soluciona el problema en Windows por el que la instalación falla cuando una ruta raíz contiene espacios. • Soluciona el problema por el que un componente que se apagaba durante una implementación utilizaba el script de apagado de la nueva versión. • Varias mejoras de apagado. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.7.0	Nuevas características • Actualiza el núcleo de Greengrass para enviar actualizaciones de estado a la AWS IoT Greengrass nube cuando el dispositivo principal aplica un despliegue local. • Añade compatibilidad con los certificados de cliente firmados por una autoridad de certificación (CA) personalizada, en la que la CA no esté registrada. AWS IoT Para utilizar esta característica, puede establecer la nueva opción de configuración `greengrassDataPlaneEndpoint` en `iotdata`. Para obtener más información, consulte Use un certificado de dispositivo firmado por una CA privada. Mejoras y correcciones de errores • Soluciona el problema por el que el núcleo de Greengrass retrasa una implementación en determinadas situaciones cuando el núcleo se detiene o se reinicia. El núcleo reanuda la implementación después de que se reinicie. • Actualiza el instalador de Greengrass para que respete el argumento `--start` cuando especifica configurar el software como un servicio del sistema. • Actualiza el comportamiento de SubscribeToComponentUpdates para establecer el ID de implementación en los casos en los que el núcleo actualiza un componente. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.6.0	Nuevas características • Añade compatibilidad con los caracteres comodín de MQTT al suscribirse a temas locales publish/subscribe . Para obtener más información, consulte Publicar/suscribir mensajes locales y SubscribeToTopic. • Suma compatibilidad con variables de receta en las configuraciones de componentes, distintas de la variable de receta *component_dependency_name* `:configuration:` *json_pointer* . Puede usar estas variables de receta al definir un componente `DefaultConfiguration` en una receta o al configurar un componente en una implementación. Para activar esta función, defina la opción de interpolateComponentConfigurationconfiguración en. `true` Para obtener más información, consulte Variables de receta y Uso de variables de receta en las actualizaciones de combinación. • Agrega una compatibilidad total con el carácter comodín * en las políticas de autorización de la comunicación entre procesos (IPC). Ahora puede especificar el carácter * de una cadena de recursos para que coincida con cualquier combinación de caracteres. Para obtener más información, consulte Comodines en las políticas de autorización. • Agrega compatibilidad con componentes personalizados para llamar a las operaciones de IPC que utiliza la CLI de Greengrass. Puede usar estas operaciones de IPC para administrar las implementaciones locales, ver los detalles de los componentes y generar una contraseña que podrá usar para iniciar sesión en la consola de depuración local. Para obtener más información, consulte IPC: administrar las implementaciones y los componentes locales. Mejoras y correcciones de errores • Soluciona el problema por el que los componentes dependientes no reaccionaban cuando sus dependencias principales se reiniciaban o cambiaban de estado en determinadas situaciones. • Mejora los mensajes de error que el dispositivo principal envía al servicio AWS IoT Greengrass en la nube cuando se produce un error en la implementación.

Versión	Cambios
	• Soluciona el problema por el que el núcleo de Greengrass aplicaba una implementación de un objeto dos veces en determinadas situaciones cuando el núcleo se reiniciaba. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
2.5.6	Nuevas características • Suma compatibilidad con los módulos de seguridad de hardware que utilizan claves ECC. Puede utilizar un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado del dispositivo. Para obtener más información, consulte Integración de la seguridad de hardware. Mejoras y correcciones de errores • Soluciona el problema por el que la implementación nunca se completa cuando se implementa un componente con un script de instalación defectuoso en determinadas situaciones. • Mejora el rendimiento durante el arranque. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.5.5	Nuevas características • Agrega la variable de entorno `GG_ROOT_CA_PATH` para los componentes, de forma que pueda acceder al certificado raíz de la autoridad de certificados (CA) raíz en los componentes personalizados. Mejoras y correcciones de errores • Suma compatibilidad con dispositivos Windows que utilizan un idioma de visualización distinto del inglés. • Actualiza la forma en que el núcleo de Greengrass analiza los argumentos booleanos del instalador, de modo que puede especificar un argumento booleano sin un valor booleano para especificar un valor `true`. Por ejemplo, ahora puede especificar `--provision` en lugar de `--provision true` para instalar con el aprovisionamiento automático de recursos. • Soluciona el problema por el que el dispositivo principal no informaba de su estado al servicio en la nube de AWS IoT Greengrass después del aprovisionamiento en determinadas situaciones. • Correcciones y mejoras menores adicionales.
2.5.4	Mejoras y correcciones de errores • Corrección de errores y mejoras generales.
2.5.3	Nuevas características • Agrega compatibilidad con la integración de seguridad de hardware. Puede utilizar un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado del dispositivo. Para obtener más información, consulte Integración de la seguridad de hardware. Mejoras y correcciones de errores • Soluciona el problema con las excepciones de tiempo de ejecución mientras el núcleo establece conexiones MQTT con AWS IoT Core.

Versión	Cambios
2.5.2	Mejoras y correcciones de errores • Soluciona el problema por el que, una vez actualizado el núcleo de Greengrass, el servicio de Windows no se puede iniciar de nuevo después de detenerlo o reiniciar el dispositivo.
2.5.1	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Agrega compatibilidad con las versiones de 32 bits del Entorno de ejecución de Java (JRE) en Windows. • Cambia el comportamiento de eliminación de grupos de objetos en los dispositivos principales cuya política AWS IoT no concede el permiso `greengrass:ListThingGroupsForCoreDevice`. Con esta versión, la implementación continúa, registra una advertencia y no elimina componentes al quitar el dispositivo principal de un grupo de objetos. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos. • Soluciona el problema con las variables de entorno del sistema que el núcleo de Greengrass pone a disposición de los procesos de los componentes de Greengrass. Ahora puede reiniciar un componente para que utilice las variables de entorno del sistema más recientes.

Versión	Cambios
2.5.0	Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Modifica el comportamiento de la eliminación de grupos de objetos. Con esta versión, puede eliminar un dispositivo principal de un grupo de objetos para desinstalar los componentes de ese grupo de objetos en la siguiente implementación. Como resultado de este cambio, la AWS IoT política de un dispositivo principal debe contar con el `greengrass:ListThingGroupsForCoreDevice` permiso. Si usó el instalador de software AWS IoT Greengrass Core para aprovisionar recursos, la AWS IoT política predeterminada lo permite `greengrass:*` , e incluye este permiso. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass. • Suma compatibilidad con las configuraciones de proxy HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red. • Suma el nuevo parámetro de configuración `windowsUser` . Puede usar este parámetro para especificar el usuario predeterminado que se utilizará para ejecutar los componentes en un dispositivo principal de Windows. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes. • Agrega las nuevas opciones de configuración `httpClient` que puede utilizar para personalizar los tiempos de espera de las solicitudes HTTP a fin de mejorar el rendimiento en redes lentas. Para obtener más información, consulte el parámetro de configuración httpClient. Mejoras y correcciones de errores • Corrige la opción de ciclo de vida de arranque para reiniciar el dispositivo principal desde un componente. • Suma compatibilidad con guiones en las variables de la receta. • Corrige la autorización de IPC para los componentes de la función de Lambda bajo demanda.

Versión	Cambios
	• Mejora los mensajes de registro y cambia los registros no críticos de `INFO` a `DEBUG`, por lo que los registros son más útiles. • Elimina el `iot:DescribeCertificate` permiso de la función de intercambio de fichas predeterminada que el núcleo de Greengrass crea al instalar el software AWS IoT Greengrass Core con aprovisionamiento automático. El núcleo de Greengrass no utiliza este permiso. • Soluciona el problema por el que el script de aprovisionamiento automático no requería el permiso `iam:GetPolicy` si `iam:CreatePolicy` estaba disponible para la misma política. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.4.0	Nuevas características • Suma compatibilidad con los límites de recursos del sistema. Puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente pueden usar en el dispositivo principal. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes. • Agrega operaciones de IPC para pausar y reanudar los component es. Para obtener más información, consulte PauseComponent y ResumeComponent. • Suma compatibilidad con el aprovisionamiento de complementos. Puede especificar un archivo JAR para que se ejecute durante la instalación a fin de aprovisionar AWS los recursos necesarios para un dispositivo principal de Greengrass. El núcleo de Greengrass incluye una interfaz que puede implementar para desarrollar complementos de aprovisionamiento personalizados. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisio namiento de recursos personalizado. • Agrega el `thing-name-policy` argumento opcional al instalador del software AWS IoT Greengrass principal. Puede usar esta opción para especificar una AWS IoT política existente o personalizada al instalar el software AWS IoT Greengrass Core con aprovisionamiento automático de recursos. Mejoras y correcciones de errores • Actualiza la configuración de registro en el arranque. Esto soluciona el problema por el que la configuración de registro no se aplicaba en el arranque. • Actualiza el enlace simbólico del cargador de núcleos para que apunte al almacén de componentes de la carpeta raíz de Greengrass durante la instalación. Esta actualización le permite eliminar el archivo JAR y otros artefactos del núcleo que se descargan al instalar el software AWS IoT Greengrass Core.

Versión	Cambios
	• Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
2.3.0	Nuevas características • Agrega compatibilidad con la implementación de documentos de configuración de hasta 10 MB, en lugar de 7 KB (para implementaciones dirigidas a objetos) o 31 KB (para implementaciones dirigidas a grupos de objetos). Para usar esta función, la AWS IoT política de un dispositivo principal debe permitir el `greengrass:GetDeploymentConfiguration` permiso. Si utilizaste el instalador del software AWS IoT Greengrass principal para aprovisionar recursos, la AWS IoT política de tu dispositivo principal lo permite `greengrass:*`, e incluye este permiso. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass. • Agrega la variable de la receta `iot:thingName`. Puedes usar esta variable de receta para obtener el nombre del dispositivo AWS IoT principal en una receta. Para obtener más información, consulte Variables de receta. Mejoras y correcciones de errores • Correcciones y mejoras menores adicionales. Para obtener más información, consulta las versiones en GitHub.
2.2.0	Nuevas características • Agrega operaciones de IPC para la administración de sombras locales. Mejoras y correcciones de errores • Reduce el tamaño del archivo JAR. • Reduce el uso de la memoria. • Soluciona problemas por los que la configuración del registro no se actualizaba en determinados casos. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.

Versión	Cambios

2.1.0	Nuevas características
	• Admite la descarga de imágenes de Docker desde repositorios privados en Amazon ECR.
	• Agrega los siguientes parámetros para personalizar la configuración de MQTT en los dispositivos principales:
	  • `maxInFlightPublishes` : el número máximo de mensajes QoS 1 de MQTT sin confirmar que pueden estar en proceso al mismo tiempo.
	  • `maxPublishRetry` : el número máximo de reintentos permitidos para un mensaje que no se publica.
	• Agrega el parámetro de configuración `fleetstatusservice` para configurar el intervalo en el que el dispositivo principal publica el estado del dispositivo en la Nube de AWS.
	• Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
	Mejoras y correcciones de errores
	• Soluciona el problema que provocaba que las implementaciones de sombras se duplicaran cuando se reiniciaba el núcleo.
	• Soluciona el problema que provocaba que el núcleo se bloqueara cuando detectaba una excepción de carga de servicio.
	• Mejora la resolución de dependencias de componentes para evitar que se produzca un error en una implementación que incluya una dependencia circular.
	• Soluciona el problema que impedía volver a implementar un component e de un complemento si este se había eliminado previamente del dispositivo principal.
	• Se corrigió el problema que provocaba que la variable de entorno `HOME` se estableciera en el directorio */greengrass/v2* `/work` de los componentes de Lambda o de los componentes que se ejecutan como raíz. La variable `HOME` ahora está correctamente configurada en el directorio principal del usuario que ejecuta el componente.

Versión	Cambios
	• Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
2.0.5	Mejoras y correcciones de errores • Al descargar los componentes AWS proporcionados, enruta correctam ente el tráfico a través de un proxy de red configurado. • Utilice el punto de conexión correcto del plano de datos de Greengrass en las regiones de AWS de China.

Versión	Cambios
2.0.4	Nuevas características • Activa el tráfico HTTPS a través del puerto 443. Puede usar el nuevo parámetro de configuración `greengrassDataPlanePort` de la versión 2.0.4 del componente de núcleo para configurar la comunicación HTTPS para que viaje por el puerto 443 en lugar del puerto predeterminado 8443. Para obtener más información, consulte Configuración de HTTPS a través del puerto 443. • Agrega la variable de receta de la ruta de trabajo. Puede utilizar esta variable de receta para obtener la ruta a las carpetas de trabajo de los componentes, que puede utilizar para compartir archivos entre los componentes y sus dependencias. Para obtener más información, consulte la variable de receta de ruta de trabajo. Mejoras y correcciones de errores • Impide la creación de la política de roles de intercambio de tokens AWS Identity and Access Management (IAM) si ya existe una política de roles. Como resultado de este cambio, el instalador ahora necesita `iam:GetPolicy` y `sts:GetCallerIdentity` cuando se ejecuta con `--provision true` . Para obtener más información, consulte Política de IAM mínima para que el instalador aprovisione recursos. • Gestiona correctamente la cancelación de una implementación que aún no se registró correctamente. • Actualiza la configuración para eliminar las entradas antiguas con marcas temporales más recientes al anular una implementación. • Correcciones y mejoras menores adicionales. Para obtener más información, consulte las versiones en GitHub.
2.0.3	Versión inicial.

Versión lite del núcleo de Greengrass

La versión lite del núcleo de Greengrass (`aws.greengrass.NucleusLite`) es un dispositivo de tiempo de ejecución para dispositivos de periferia restringidos optimizado para ocupar una memoria mínima (utiliza menos de 5 MB de RAM). Se introdujo con la AWS IoT Greengrass versión 2.14.0 y es retrocompatible con componentes AWS IoT Greengrass genéricos, la API Greengrass V2 y el SDK.

La versión lite del núcleo de Greengrass se ofrece como una alternativa al conocido núcleo de Greengrass (`aws.greengrass.Nucleus`) y se puede utilizar en flotas heterogéneas de dispositivos Greengrass.

Temas

- Versiones
- Sistema operativo
- Requisitos
- Compatibilidad
- Descarga e instalación
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux (distribuciones con systemd)

Para obtener más información, consulte núcleo de Greengrass.

Requisitos

Los dispositivos deben cumplir ciertos requisitos para instalar y ejecutar el software AWS IoT Greengrass nucleus lite y Core. AWS IoT Greengrass Para obtener más información, consulte la Guía de configuración.

- 5 MB de espacio de RAM para el tiempo de ejecución del núcleo.
- 5 MB de almacenamiento (disco/flash).

Las dependencias adicionales del sistema se documentan en la Guía de configuración.

Se admite la ejecución del componente núcleo de Greengrass en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente:

- El núcleo de Greengrass debe tener conectividad con los AWS IoT datos, AWS IoT las credenciales y Amazon S3.

Compatibilidad

El AWS IoT Greengrass núcleo lite es compatible con la API AWS IoT Greengrass v2 (subconjunto de) y es compatible. SDKs No depende de ningún lenguaje específico, runtimes/VMs pero los componentes que se agregan a una implementación pueden requerir la presencia de tiempos de ejecución específicos (por ejemplo, Java, JVM, Python). Para obtener más información sobre las características compatibles con la versión lite del núcleo de Greengrass, consulte Compatibilidad de características de Greengrass.

Descarga e instalación

Puede descargar un paquete apt, compilarlo desde el código fuente, usar una capa de Yocto o descargar una imagen de Yocto prediseñada para un dispositivo compatible (por ejemplo,). RaspberryPi Desde la Consola de AWS IoT Core, podrá descargar un kit de conexión que contiene todas las credenciales y la configuración inicial de su dispositivo. Las instrucciones sobre cómo realizar la instalación se incluyen en cada método de distribución específico.

También puede seguir un tutorial para instalar el AWS IoT Greengrass núcleo lite y explorar el desarrollo de componentes de Greengrass. Para obtener más información, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

Configuración

El núcleo ofrece los siguientes parámetros de configuración: Algunos parámetros requieren que el software AWS IoT Greengrass Core se reinicie para que surta efecto.

`iotRoleAlias`

El alias del AWS IoT rol que apunta a un rol de IAM de intercambio de fichas. El proveedor de AWS IoT credenciales asume esta función para permitir que el dispositivo principal de Greengrass interactúe con AWS los servicios. Para obtener más información, consulte Autorizar los dispositivos principales para que interactúen con AWS los servicios.

`iotDataEndpoint`

El punto final de AWS IoT datos para su Cuenta de AWS.

`iotCredEndpoint`

El punto final de AWS IoT credenciales para su Cuenta de AWS.

`greengrassDataPlanePort`

El puerto que se utilizará para las conexiones del plano de datos. Para obtener más información, consulte Conección en el puerto 443 o a través de un proxy de red.

> **Important**
>
> Debe especificar un puerto en el que el dispositivo pueda realizar conexiones salientes. Si especificas un puerto que está bloqueado, el dispositivo no podrá conectarse AWS IoT Greengrass para recibir despliegues. Puede elegir entre las siguientes opciones:
>
> - `443`
> - `8443`
> - Valor predeterminado: `8443`

`awsRegion`

El que se Región de AWS debe usar.

`runWithDefault`

El usuario del sistema que se utilizará para ejecutar los componentes.

> **Important**
>
> Al implementar un cambio en este parámetro de configuración, el software AWS IoT Greengrass principal se reinicia para que el cambio surta efecto.

Este objeto contiene la siguiente información:

`posixUser`

El nombre o ID del usuario del sistema y, opcionalmente, del grupo de sistemas que el dispositivo principal utiliza para ejecutar los componentes genéricos. Especifique el usuario y el grupo separados por dos puntos (`:`) con el siguiente formato: `user:group`. El grupo es opcional. Si no especifica un grupo, el software AWS IoT Greengrass Core utiliza el grupo principal para el usuario. Por ejemplo, puede especificar `ggc_user` o `ggc_user:ggc_group`. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

`networkProxy`

(Opcional) El proxy de red que se utilizará en todas las conexiones. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

> **Important**
>
> Al implementar un cambio en este parámetro de configuración, el cambio se efectuará después del siguiente reinicio del software AWS IoT Greengrass principal.

Este objeto contiene la siguiente información:

`noProxyAddresses`

(Opcional) Una lista separada por comas de direcciones IP o nombres de host que están exentos del proxy.

`proxy`

El proxy al que conectar. Este objeto contiene la siguiente información:

`url`

La dirección URL del servidor proxy, en el formato `http://host:port`.

- `scheme`: el esquema, que debe ser `http`.
- `host`: el nombre de host o dirección IP del servidor proxy.
- `port`: (opcional) el número de puerto. Si no especifica el puerto, el dispositivo principal de Greengrass usará el siguiente valor predeterminado:
  - `http`: 80

Archivo de registro local

Los mensajes se registran en stdout, y systemd gestiona los archivos de registro.

Visualización de los registros de este componente

- Utilice `journalctl` para ver los registros.

Registros de cambios

Versión	Cambios
2.3.2	Mejoras y correcciones de errores • Actualiza el archivo de versiones para informar correctamente sobre la versión de Nucleus.
2.3.1	Mejoras y correcciones de errores • GG no intentará actualizar el estado de despliegue de los trabajos cancelados. • Asegúrese de que el DIR de fdopendir esté cerrado. • El aprovisionamiento de flotas ahora activará la sobrescritura del archivo de certificado en cada ejecución. • Otras correcciones de errores menores.
2.3.0	Nuevas características • Support para usar TPM 2.0 para la autorización MQTT de IoT Core.

Versión	Cambios
	• Los paquetes apt de muestra ahora son compatibles con más sistemas operativos: Ubuntu 22.04, Ubuntu 24.04, Debian 12 y Debian 13. • RestartComponent Ahora se admite IPC. Mejoras y correcciones de errores • Las implementaciones locales ya no necesitan acceso a Internet. • GetConfiguration se ha actualizado para que coincida con el comportam iento en tiempo de ejecución de Greengrass Nucleus. (Un cambio radical) • Corrección de errores y mejoras generales.
2.2.2	Mejoras y correcciones de errores • Soluciona las implementaciones revisadas que contenían versiones de componentes sin cambios que fallaban cuando un componente no modificado tenía un archivo ejecutable en ejecución como artefacto. • Soluciona la interpolación de las variables de receta cuando el valor interpolado tiene más de cuatro subobjetos anidados. • Soluciona la interpolación de variables de receta, como las comillas y los caracteres especiales del intérprete de comandos. • Soluciona la pérdida de fds cuando fallan las conexiones MQTT.
2.2.1	Mejoras y correcciones de errores • Soluciona el problema que impedía que el núcleo obtuviera las credenciales de TES.
2.2.0	Nuevas características • Añade compatibilidad con el artefacto URIs de imagen del contenedor. Mejoras y correcciones de errores • Corrección de errores y mejoras generales.

Versión	Cambios
2.1.0	Nuevas características • Agrega compatibilidad con el proxy HTTP, que se puede configurar mediante la opción de configuración `networkProxy` . Mejoras y correcciones de errores • Reduce el requisito de `libcurl` desde 7,86 a 7,82 para adaptarse a dispositivos que ejecutan versiones `libcurl` anteriores. • Actualiza los registros `journalctl` para que se atribuyan a los componentes en lugar de a recipe-runner. • Mejora las respuestas de error en las llamadas de IPC. • Agrega reintentos para los intentos de descarga de S3 para artefactos de componentes genéricos. • Correcciones de errores menores.
2.0.2	Mejoras y correcciones de errores • Corrige las dependencias de los paquetes apt para incluir `cgroup-tools` .
2.0.1	Mejoras y correcciones de errores • Agrega compatibilidad con la interpolación de variables de receta para la sección de tiempo de espera de la receta de Greengrass. • Añade compatibilidad con el comando ValidateAuthorizationToken IPC para el administrador de flujos. • Soluciona las advertencias del aprovisionamiento de la flota. • Agrega el reintento y retraso al oyente de tareas. • Corrección de errores y mejoras generales.
2.0.0	Versión inicial.

Autenticación del dispositivo de cliente

El componente de autenticación del dispositivo de cliente (`aws.greengrass.clientdevices.Auth`) autentica los dispositivos de cliente y autoriza las acciones de los dispositivos de cliente.

> **Note**
>
> Los dispositivos de cliente son dispositivos IoT locales que se conectan a un dispositivo principal de Greengrass para enviar mensajes MQTT y datos para su procesamiento. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

> **Note**
>
> Se ha dejado de usar la versión 2.3.0 de autenticación del dispositivo de cliente. Se recomienda encarecidamente que actualice a la versión 2.3.1 o posterior de autenticación del dispositivo de cliente.

Este componente tiene las siguientes versiones:

- 2.5.x

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El rol de servicio de Greengrass debe estar asociado a usted Cuenta de AWS y permitir el `iot:DescribeCertificate` permiso.
- La AWS IoT política del dispositivo principal debe permitir los siguientes permisos:
  - `greengrass:GetConnectivityInfo`, donde los recursos incluyen el ARN del dispositivo principal que ejecuta este componente
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`, donde los recursos incluyen el nombre de recurso de Amazon (ARN) de cada dispositivo de cliente que se conecta al dispositivo principal

  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:PutCertificateAuthorities`
  - `iot:Publish`, donde los recursos incluyen el ARN del siguiente tema de MQTT:
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get`
  - `iot:Subscribe`, donde los recursos incluyen los siguientes filtros ARNs de temas de MQTT:
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/update/delta`
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get/accepted`
  - `iot:Receive`, donde los recursos incluyen los siguientes ARNs temas de MQTT:
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/update/delta`
    - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get/accepted`

  Para obtener más información, consulte AWS IoT políticas para las operaciones del plano de datos y AWS IoT Política mínima de compatibilidad con los dispositivos cliente.
- (Opcional) Para utilizar la autenticación sin conexión, la función AWS Identity and Access Management (IAM) utilizada por el AWS IoT Greengrass servicio debe contener el siguiente permiso:
  - `greengrass:ListClientDevicesAssociatedWithCoreDevice` para permitir que el dispositivo principal enumere los clientes para la autenticación sin conexión.
- Se admite la ejecución del componente de autenticación del dispositivo de cliente en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.
  - El componente de autenticación del dispositivo cliente debe tener conectividad con AWS IoT data AWS IoT Credentials y Amazon S3.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`iot.region.amazonaws.com`	443	Sí	Se utiliza para obtener información sobre los certificados de AWS IoT cosas.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.5.5

En la siguiente tabla se enumeran las dependencias de la versión 2.5.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.17.0	Flexible

2.5.4

La siguiente tabla muestra las dependencias de la versión 2.5.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.16.0	Flexible

2.5.2 – 2.5.3

La siguiente tabla muestra las dependencias de las versiones 2.5.2 y 2.5.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.15.0	Flexible

2.5.1

En la siguiente tabla, se muestran las dependencias de la versión 2.5.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.14.0	Flexible

2.4.4 - 2.5.0

En la siguiente tabla, se muestran las dependencias de la versión 2.4.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.13.0	Flexible

2.4.3

En la siguiente tabla, se muestran las dependencias de la versión 2.4.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.12.0	Flexible

2.4.1 and 2.4.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.4.1 y 2.4.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.11.0	Flexible

2.3.0 – 2.4.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.0 a 2.4.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.10.0	Flexible

2.3.0

En la siguiente tabla, se muestran las dependencias de la versión 2.3.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.10.0	Flexible

2.2.3

En la siguiente tabla, se muestran las dependencias de la versión 2.2.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <=2.9.0	Flexible

2.2.2

En la siguiente tabla, se muestran las dependencias de la versión 2.2.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <=2.8.0	Flexible

2.2.1

En la siguiente tabla, se muestran las dependencias de la versión 2.2.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.8.0	Flexible

2.2.0

En la siguiente tabla, se muestran las dependencias de la versión 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.6.0 <2.7.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.7.0	Flexible

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.6.0	Flexible

2.0.2 and 2.0.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.2 y 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.5.0	Flexible

2.0.1

En la siguiente tabla, se muestran las dependencias de la versión 2.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.4.0	Flexible

2.0.0

En la siguiente tabla, se muestran las dependencias de la versión 2.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.3.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

> **Note**
>
> El permiso de suscripción se evalúa durante una solicitud de suscripción del cliente al agente MQTT local. Si se revoca el permiso de suscripción actual del cliente, el cliente ya no podrá

suscribirse a un tema. Sin embargo, seguirá recibiendo mensajes de cualquier tema al que se haya suscrito anteriormente. Para evitar este comportamiento, el agente MQTT local debe reiniciarse tras revocar el permiso de suscripción para forzar la reautorización de los clientes. Para el componente agente MQTT 5 (EMQX), actualice la configuración `restartIdentifier` para reiniciar el agente MQTT 5.
Para el componente agente MQTT 3.1.1 (Moquette), se reinicia semanalmente de forma predeterminada cuando el certificado del servidor cambia y obliga a los clientes a volver a autorizar. Puede forzar un reinicio si cambia la información de conectividad (direcciones IP) del dispositivo principal o realiza una implementación para eliminar el componente agente y volver a implementarlo más adelante.

v2.5.0 – 2.5.4

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

groupNameKey

El nombre de este grupo de dispositivos. *groupNameKey*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al principio y al final del nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen o terminen con la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`— El nombre del dispositivo de un cliente AWS IoT .

  Example Ejemplo de regla de selección

  La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

  ```
  thingName: MyClientDevice1 OR thingName: MyClientDevice2
  ```

  Example Ejemplo de regla de selección (usar caracteres comodín)

  La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

  ```
  thingName: MyClientDevice*
  ```

  Example Ejemplo de regla de selección (usar caracteres comodín)

  La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres terminan en `MyClientDevice`.

  ```
  thingName: *MyClientDevice
  ```

  Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

  La siguiente regla de selección coincide con todos los dispositivos de cliente.

  ```
  thingName: *
  ```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

policyNameKey

El nombre de esta política de autorización. *policyNameKey*Sustitúyalo por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

statementNameKey

El nombre de esta declaración de política. *statementNameKey*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

  Esta operación admite los siguientes recursos:

  - `mqtt:clientId:`*deviceClientId*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *deviceClientId*Sustitúyalo por el ID de cliente que se va a utilizar.
- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

  - `mqtt:topic:`*mqttTopic*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *mqttTopic*Sustitúyalo por el tema que se va a utilizar.

    Este recurso no admite caracteres comodín de temas MQTT.
- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

- `mqtt:topicfilter:`*`mqttTopicFilter`*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *`mqttTopicFilter`*Sustitúyalo por el filtro de tema que desee utilizar.

  Este recurso no admite caracteres comodín de temas MQTT.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín `*` en cualquier lugar de la variable de recurso para permitir el acceso a todos los recursos. Por ejemplo, puede especificar **`mqtt:topic:my*`** para permitir el acceso a los recursos que coincidan con esa entrada.

Se admite la siguiente variable de recurso:

- `mqtt:topic:${iot:Connection.Thing.ThingName}`

  Esto se traduce en el nombre del elemento del AWS IoT Core registro para el que se está evaluando la política. AWS IoT Core usa el certificado que presenta el dispositivo cuando se autentica para determinar qué se debe usar para verificar la conexión. Esta variable de política solo está disponible cuando un dispositivo se conecta a través de MQTT o MQTT a través del protocolo. WebSocket

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con

la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

`performance`

(Opcional) Las opciones de configuración del rendimiento de este dispositivo principal. Este objeto contiene la siguiente información:

`maxActiveAuthTokens`

(Opcional) El número máximo de tokens de autorización de dispositivos de cliente activos. Puede aumentar este número para permitir que un mayor número de dispositivos de cliente se conecten a un dispositivo principal sin tener que volver a autenticarlos.

Valor predeterminado: `2500`

`cloudRequestQueueSize`

(Opcional) El número máximo de Nube de AWS solicitudes que se deben poner en cola antes de que este componente las rechace.

Valor predeterminado: `100`

`maxConcurrentCloudRequests`

(Opcional) El número máximo de solicitudes simultáneas que se enviarán a la Nube de AWS. Puede aumentar este número para mejorar el rendimiento de la autenticación en los dispositivos principales a los que se conecta un gran número de dispositivos de cliente.

Valor predeterminado: `1`

`certificateAuthority`

(Opcional) Opciones de configuración de la autoridad de certificación para reemplazar la autoridad intermedia del dispositivo principal por su propia autoridad de certificación intermedia.

> **Note**
>
> Si configura su dispositivo principal de Greengrass con una autoridad de certificación (CA) personalizada y usa la misma CA para emitir los certificados de los dispositivos de cliente, Greengrass omite las comprobaciones de las políticas de autorización para las operaciones MQTT de los dispositivos de cliente. El componente de autenticación del dispositivo de cliente confía plenamente en los clientes que usan certificados firmados por la CA para la que está configurado.
> Para restringir este comportamiento al usar una CA personalizada, cree y firme los dispositivos de cliente con una CA diferente o intermedia y, a continuación, ajuste los campos `certificateUri` y `certificateChainUri` para que apunten a la CA intermedia correcta.

Este objeto contiene la siguiente información:

certificateUri

La ubicación del certificado. Puede ser un URI del sistema de archivos o un URI que apunta a un certificado almacenado en un módulo de seguridad de hardware.

`certificateChainUri`

La ubicación de la cadena de certificados para la CA del dispositivo principal. Debe ser la cadena de certificados completa que lleva a la CA raíz. Puede ser un URI del sistema de archivos o un URI que apunta a una cadena de certificados almacenada en un módulo de seguridad de hardware.

`privateKeyUri`

La ubicación de la clave privada del dispositivo principal. Puede ser un URI del sistema de archivos o un URI que apunta a una clave privada de certificado almacenada en un módulo de seguridad de hardware.

`security`

(Opcional) Opciones de configuración de seguridad para este dispositivo principal. Este objeto contiene la siguiente información:

`clientDeviceTrustDurationMinutes`

El tiempo en minutos durante el que se puede confiar en la información de autenticación de un dispositivo de cliente antes de que sea necesario volver a autenticarse con el dispositivo principal. El valor predeterminado es 1.

`metrics`

(Opcional) Las opciones de métricas de este dispositivo principal. Las métricas de error solo se mostrarán si hay un error en la autenticación del dispositivo de cliente. Este objeto contiene la siguiente información:

`disableMetrics`

Si el campo `disableMetrics` está establecido como `true`, la autenticación del dispositivo de cliente no recopilará métricas.

Valor predeterminado: `false`

`aggregatePeriodSeconds`

El periodo de agregación en segundos que determina la frecuencia con la que la autenticación del dispositivo de cliente agrega las métricas y las envía al agente de telemetría. Esto no cambia la frecuencia con la que se publican las métricas, ya que el agente de telemetría sigue publicándolas una vez al día.

Valor predeterminado: `3600`

startupTimeoutSeconds

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `ERRORED` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

En el siguiente ejemplo de configuración se especifica que se permita la conexión de los dispositivos cliente cuyos nombres comiencen por «» y publish/subscribe en todos los temas. `MyClientDevice`

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
```

```
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política de nombres de objetos)

El siguiente ejemplo de configuración permite a los dispositivos de cliente publicar temas que comiencen con el nombre del objeto del dispositivo de cliente y terminen con la cadena `topic`.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "myThing": {
        "selectionRule": "thingName: *",
        "policyName": "MyThingNamePolicy"
      }
    },
    "policies": {
      "MyThingNamePolicy": {
        "policyStatement": {
          "statementDescription": "mqtt publish",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:${iot:Connection.Thing.ThingName}/*/topic"
          ]
        }
      }
    }
  }
}
```

v2.4.5

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

`groupNameKey`

El nombre de este grupo de dispositivos. *`groupNameKey`*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al principio y al final del

nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen o terminen con la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`— El nombre del dispositivo de un cliente AWS IoT .

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres terminan en `MyClientDevice`.

```
thingName: *MyClientDevice
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

`policyNameKey`

El nombre de esta política de autorización. *`policyNameKey`*Sustitúyalo por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

`statementNameKey`

El nombre de esta declaración de política. *`statementNameKey`*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

Esta operación admite los siguientes recursos:

- `mqtt:clientId:`*`deviceClientId`*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *`deviceClientId`*Sustitúyalo por el ID de cliente que se va a utilizar.

- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

  - `mqtt:topic:`*`mqttTopic`*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *`mqttTopic`*Sustitúyalo por el tema que se va a utilizar.

    Este recurso no admite caracteres comodín de temas MQTT.

- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *`mqttTopicFilter`*Sustitúyalo por el filtro de tema que desee utilizar.

    Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

    El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta

política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín `*` para permitir el acceso a todos los recursos. No puede usar el comodín `*` para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

`performance`

(Opcional) Las opciones de configuración del rendimiento de este dispositivo principal. Este objeto contiene la siguiente información:

`maxActiveAuthTokens`

(Opcional) El número máximo de tokens de autorización de dispositivos de cliente activos. Puede aumentar este número para permitir que un mayor número de dispositivos de cliente se conecten a un dispositivo principal sin tener que volver a autenticarlos.

Valor predeterminado: `2500`

`cloudRequestQueueSize`

(Opcional) El número máximo de Nube de AWS solicitudes que se van a poner en cola antes de que este componente las rechace.

Valor predeterminado: `100`

`maxConcurrentCloudRequests`

(Opcional) El número máximo de solicitudes simultáneas que se enviarán a la Nube de AWS. Puede aumentar este número para mejorar el rendimiento de la autenticación en los dispositivos principales a los que se conecta un gran número de dispositivos de cliente.

Valor predeterminado: `1`

`certificateAuthority`

(Opcional) Opciones de configuración de la autoridad de certificación para reemplazar la autoridad intermedia del dispositivo principal por su propia autoridad de certificación intermedia.

> **Note**
>
> Si configura su dispositivo principal de Greengrass con una autoridad de certificación (CA) personalizada y usa la misma CA para emitir los certificados de los dispositivos de cliente, Greengrass omite las comprobaciones de las políticas de autorización para las operaciones MQTT de los dispositivos de cliente. El componente de autenticación del dispositivo de cliente confía plenamente en los clientes que usan certificados firmados por la CA para la que está configurado.
> Para restringir este comportamiento al usar una CA personalizada, cree y firme los dispositivos de cliente con una CA diferente o intermedia y, a continuación, ajuste los campos `certificateUri` y `certificateChainUri` para que apunten a la CA intermedia correcta.

Este objeto contiene la siguiente información:

certificateUri

La ubicación del certificado. Puede ser un URI del sistema de archivos o un URI que apunta a un certificado almacenado en un módulo de seguridad de hardware.

`certificateChainUri`

La ubicación de la cadena de certificados para la CA del dispositivo principal. Debe ser la cadena de certificados completa que lleva a la CA raíz. Puede ser un URI del sistema de archivos o un URI que apunta a una cadena de certificados almacenada en un módulo de seguridad de hardware.

`privateKeyUri`

La ubicación de la clave privada del dispositivo principal. Puede ser un URI del sistema de archivos o un URI que apunta a una clave privada de certificado almacenada en un módulo de seguridad de hardware.

`security`

(Opcional) Opciones de configuración de seguridad para este dispositivo principal. Este objeto contiene la siguiente información:

`clientDeviceTrustDurationMinutes`

El tiempo en minutos durante el que se puede confiar en la información de autenticación de un dispositivo de cliente antes de que sea necesario volver a autenticarse con el dispositivo principal. El valor predeterminado es 1.

`metrics`

(Opcional) Las opciones de métricas de este dispositivo principal. Las métricas de error solo se mostrarán si hay un error en la autenticación del dispositivo de cliente. Este objeto contiene la siguiente información:

`disableMetrics`

Si el campo `disableMetrics` está establecido como `true`, la autenticación del dispositivo de cliente no recopilará métricas.

Valor predeterminado: `false`

`aggregatePeriodSeconds`

El periodo de agregación en segundos que determina la frecuencia con la que la autenticación del dispositivo de cliente agrega las métricas y las envía al agente de telemetría. Esto no cambia la frecuencia con la que se publican las métricas, ya que el agente de telemetría sigue publicándolas una vez al día.

Valor predeterminado: `3600`

startupTimeoutSeconds

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `ERRORED` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

En el siguiente ejemplo de configuración se especifica que se permita la conexión de los dispositivos cliente cuyos nombres comiencen por «» y publish/subscribe en todos los temas. `MyClientDevice`

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
```

```
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
```

```
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.2 - v2.4.4

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

groupNameKey

El nombre de este grupo de dispositivos. *groupNameKey*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al final del nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen por la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`: el nombre del objeto de AWS IoT del dispositivo de cliente.

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

`policyNameKey`

El nombre de esta política de autorización. *`policyNameKey`*Sustitúyala por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

statementNameKey

El nombre de esta declaración de política. *statementNameKey*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

  Esta operación admite los siguientes recursos:

  - `mqtt:clientId:`*deviceClientId*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *deviceClientId*Sustitúyalo por el ID de cliente que se va a utilizar.
- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

  - `mqtt:topic:`*mqttTopic*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *mqttTopic*Sustitúyalo por el tema que se va a utilizar.

    Este recurso no admite caracteres comodín de temas MQTT.
- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

  - `mqtt:topicfilter:`*mqttTopicFilter*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *mqttTopicFilter*Sustitúyalo por el filtro de tema que desee utilizar.

Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín * para permitir el acceso a todos los recursos. No puede usar el comodín * para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

`performance`

(Opcional) Las opciones de configuración del rendimiento de este dispositivo principal. Este objeto contiene la siguiente información:

`maxActiveAuthTokens`

(Opcional) El número máximo de tokens de autorización de dispositivos de cliente activos. Puede aumentar este número para permitir que un mayor número de dispositivos de cliente se conecten a un dispositivo principal sin tener que volver a autenticarlos.

Valor predeterminado: `2500`

`cloudRequestQueueSize`

(Opcional) El número máximo de Nube de AWS solicitudes que se van a poner en cola antes de que este componente las rechace.

Valor predeterminado: `100`

`maxConcurrentCloudRequests`

(Opcional) El número máximo de solicitudes simultáneas que se enviarán a la Nube de AWS. Puede aumentar este número para mejorar el rendimiento de la autenticación en los dispositivos principales a los que se conecta un gran número de dispositivos de cliente.

Valor predeterminado: `1`

`certificateAuthority`

(Opcional) Opciones de configuración de la autoridad de certificación para reemplazar la autoridad intermedia del dispositivo principal por su propia autoridad de certificación intermedia.

> **Note**
>
> Si configura su dispositivo principal de Greengrass con una autoridad de certificación (CA) personalizada y usa la misma CA para emitir los certificados de los dispositivos de cliente, Greengrass omite las comprobaciones de las políticas de autorización para las operaciones MQTT de los dispositivos de cliente. El componente de autenticación del dispositivo de cliente confía plenamente en los clientes que usan certificados firmados por la CA para la que está configurado.
> Para restringir este comportamiento al usar una CA personalizada, cree y firme los dispositivos de cliente con una CA diferente o intermedia y, a continuación, ajuste los campos `certificateUri` y `certificateChainUri` para que apunten a la CA intermedia correcta.

Este objeto contiene la siguiente información:

certificateUri

La ubicación del certificado. Puede ser un URI del sistema de archivos o un URI que apunta a un certificado almacenado en un módulo de seguridad de hardware.

`certificateChainUri`

La ubicación de la cadena de certificados para la CA del dispositivo principal. Debe ser la cadena de certificados completa que lleva a la CA raíz. Puede ser un URI del sistema de archivos o un URI que apunta a una cadena de certificados almacenada en un módulo de seguridad de hardware.

`privateKeyUri`

La ubicación de la clave privada del dispositivo principal. Puede ser un URI del sistema de archivos o un URI que apunta a una clave privada de certificado almacenada en un módulo de seguridad de hardware.

`security`

(Opcional) Opciones de configuración de seguridad para este dispositivo principal. Este objeto contiene la siguiente información:

`clientDeviceTrustDurationMinutes`

El tiempo en minutos durante el que se puede confiar en la información de autenticación de un dispositivo de cliente antes de que sea necesario volver a autenticarse con el dispositivo principal. El valor predeterminado es 1.

`metrics`

(Opcional) Las opciones de métricas de este dispositivo principal. Las métricas de error solo se mostrarán si hay un error en la autenticación del dispositivo de cliente. Este objeto contiene la siguiente información:

`disableMetrics`

Si el campo `disableMetrics` está establecido como `true`, la autenticación del dispositivo de cliente no recopilará métricas.

Valor predeterminado: `false`

`aggregatePeriodSeconds`

El periodo de agregación en segundos que determina la frecuencia con la que la autenticación del dispositivo de cliente agrega las métricas y las envía al agente de telemetría. Esto no cambia la frecuencia con la que se publican las métricas, ya que el agente de telemetría sigue publicándolas una vez al día.

Valor predeterminado: `3600`

startupTimeoutSeconds

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `ERRORED` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

En el siguiente ejemplo de configuración se especifica que se permita la conexión de los dispositivos cliente cuyos nombres comiencen por «» y publish/subscribe en todos los temas. `MyClientDevice`

```
{
```

```
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

`groupNameKey`

El nombre de este grupo de dispositivos. *`groupNameKey`*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al final del nombre del

objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen por la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`: el nombre del objeto de AWS IoT del dispositivo de cliente.

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

`policyNameKey`

El nombre de esta política de autorización. *`policyNameKey`*Sustitúyala por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

`statementNameKey`

El nombre de esta declaración de política. *`statementNameKey`*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

  Esta operación admite los siguientes recursos:

  - `mqtt:clientId:`*`deviceClientId`*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *`deviceClientId`*Sustitúyalo por el ID de cliente que se va a utilizar.
- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

Esta operación admite los siguientes recursos:

- `mqtt:topic:`*`mqttTopic`*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *`mqttTopic`*Sustitúyalo por el tema que se va a utilizar.

  Este recurso no admite caracteres comodín de temas MQTT.

- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *`mqttTopicFilter`*Sustitúyalo por el filtro de tema que desee utilizar.

    Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

    El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín * para permitir el acceso a todos los recursos. No puede usar el comodín * para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

`performance`

(Opcional) Las opciones de configuración del rendimiento de este dispositivo principal. Este objeto contiene la siguiente información:

`maxActiveAuthTokens`

(Opcional) El número máximo de tokens de autorización de dispositivos de cliente activos. Puede aumentar este número para permitir que un mayor número de dispositivos de cliente se conecten a un dispositivo principal sin tener que volver a autenticarlos.

Valor predeterminado: `2500`

`cloudRequestQueueSize`

(Opcional) El número máximo de Nube de AWS solicitudes que se van a poner en cola antes de que este componente las rechace.

Valor predeterminado: `100`

`maxConcurrentCloudRequests`

(Opcional) El número máximo de solicitudes simultáneas que se enviarán a la Nube de AWS. Puede aumentar este número para mejorar el rendimiento de la autenticación en los dispositivos principales a los que se conecta un gran número de dispositivos de cliente.

Valor predeterminado: `1`

`certificateAuthority`

(Opcional) Opciones de configuración de la autoridad de certificación para reemplazar la autoridad intermedia del dispositivo principal por su propia autoridad de certificación intermedia. Este objeto contiene la siguiente información:

Este objeto contiene la siguiente información:

certificateUri

La ubicación del certificado. Puede ser un URI del sistema de archivos o un URI que apunta a un certificado almacenado en un módulo de seguridad de hardware.

`certificateChainUri`

La ubicación de la cadena de certificados para la CA del dispositivo principal. Debe ser la cadena de certificados completa que lleva a la CA raíz. Puede ser un URI del sistema de archivos o un URI que apunta a una cadena de certificados almacenada en un módulo de seguridad de hardware.

`privateKeyUri`

La ubicación de la clave privada del dispositivo principal. Puede ser un URI del sistema de archivos o un URI que apunta a una clave privada de certificado almacenada en un módulo de seguridad de hardware.

`security`

(Opcional) Opciones de configuración de seguridad para este dispositivo principal. Este objeto contiene la siguiente información:

`clientDeviceTrustDurationMinutes`

El tiempo en minutos durante el que se puede confiar en la información de autenticación de un dispositivo de cliente antes de que sea necesario volver a autenticarse con el dispositivo principal. El valor predeterminado es 1.

`metrics`

(Opcional) Las opciones de métricas de este dispositivo principal. Las métricas de error solo se mostrarán si hay un error en la autenticación del dispositivo de cliente. Este objeto contiene la siguiente información:

`disableMetrics`

Si el campo `disableMetrics` está establecido como `true`, la autenticación del dispositivo de cliente no recopilará métricas.

Valor predeterminado: `false`

`aggregatePeriodSeconds`

El periodo de agregación en segundos que determina la frecuencia con la que la autenticación del dispositivo de cliente agrega las métricas y las envía al agente de telemetría. Esto no cambia la frecuencia con la que se publican las métricas, ya que el agente de telemetría sigue publicándolas una vez al día.

Valor predeterminado: `3600`

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

En el siguiente ejemplo de configuración se especifica que se permita la conexión de los dispositivos cliente cuyos nombres comiencen por «» y publish/subscribe en todos los temas. `MyClientDevice`

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
```

```
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.3.x

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

`groupNameKey`

El nombre de este grupo de dispositivos. *`groupNameKey`*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al final del nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen por la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`: el nombre del objeto de AWS IoT del dispositivo de cliente.

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

`policyNameKey`

El nombre de esta política de autorización. *`policyNameKey`*Sustitúyala por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

`statementNameKey`

El nombre de esta declaración de política. *`statementNameKey`*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

  Esta operación admite los siguientes recursos:

  - `mqtt:clientId:`*`deviceClientId`*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *`deviceClientId`*Sustitúyalo por el ID de cliente que se va a utilizar.
- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

- `mqtt:topic:`*`mqttTopic`*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *`mqttTopic`*Sustitúyalo por el tema que se va a utilizar.

  Este recurso no admite caracteres comodín de temas MQTT.

- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *`mqttTopicFilter`*Sustitúyalo por el filtro de tema que desee utilizar.

    Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

    El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín * para permitir el acceso a todos los recursos. No puede usar el comodín * para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

`performance`

(Opcional) Las opciones de configuración del rendimiento de este dispositivo principal. Este objeto contiene la siguiente información:

`maxActiveAuthTokens`

(Opcional) El número máximo de tokens de autorización de dispositivos de cliente activos. Puede aumentar este número para permitir que un mayor número de dispositivos de cliente se conecten a un solo dispositivo principal sin volver a autenticarlos.

Valor predeterminado: `2500`

`cloudRequestQueueSize`

(Opcional) El número máximo de Nube de AWS solicitudes que se van a poner en cola antes de que este componente las rechace.

Valor predeterminado: `100`

`maxConcurrentCloudRequests`

(Opcional) El número máximo de solicitudes simultáneas que se enviarán a la Nube de AWS. Puede aumentar este número para mejorar el rendimiento de la autenticación en los dispositivos principales a los que se conecta un gran número de dispositivos de cliente.

Valor predeterminado: `1`

`certificateAuthority`

(Opcional) Opciones de configuración de la autoridad de certificación para reemplazar la autoridad intermedia del dispositivo principal por su propia autoridad de certificación intermedia. Este objeto contiene la siguiente información:

certificateUri

La ubicación del certificado. Puede ser un URI del sistema de archivos o un URI que apunta a un certificado almacenado en un módulo de seguridad de hardware.

`certificateChainUri`

La ubicación de la cadena de certificados para la CA del dispositivo principal. Debe ser la cadena de certificados completa que lleva a la CA raíz. Puede ser un URI del sistema de archivos o un URI que apunta a una cadena de certificados almacenada en un módulo de seguridad de hardware.

`privateKeyUri`

La ubicación de la clave privada del dispositivo principal. Puede ser un URI del sistema de archivos o un URI que apunta a una clave privada de certificado almacenada en un módulo de seguridad de hardware.

`security`

(Opcional) Opciones de configuración de seguridad para este dispositivo principal. Este objeto contiene la siguiente información:

`clientDeviceTrustDurationMinutes`

La duración en minutos durante la que se puede confiar en la información de autenticación de un dispositivo de cliente antes de que sea necesario volver a autenticarse con el dispositivo principal. El valor predeterminado es 1.

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

En el siguiente ejemplo de configuración se especifica que se permita la conexión de los dispositivos cliente cuyos nombres comiencen por «» y publish/subscribe en todos los temas. `MyClientDevice`

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
```

```
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.2.x

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

`groupNameKey`

El nombre de este grupo de dispositivos. *`groupNameKey`*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al final del nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen por la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`: el nombre del objeto de AWS IoT del dispositivo de cliente.

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

`policyNameKey`

El nombre de esta política de autorización. *`policyNameKey`*Sustitúyala por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

`statementNameKey`

El nombre de esta declaración de política. *`statementNameKey`*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

Esta operación admite los siguientes recursos:

- `mqtt:clientId:`*`deviceClientId`*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *`deviceClientId`*Sustitúyalo por el ID de cliente que se va a utilizar.

- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

  - `mqtt:topic:`*`mqttTopic`*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *`mqttTopic`*Sustitúyalo por el tema que se va a utilizar.

    Este recurso no admite caracteres comodín de temas MQTT.

- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *`mqttTopicFilter`*Sustitúyalo por el filtro de tema que desee utilizar.

    Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

    El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta

política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín `*` para permitir el acceso a todos los recursos. No puede usar el comodín `*` para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

`performance`

(Opcional) Las opciones de configuración del rendimiento de este dispositivo principal. Este objeto contiene la siguiente información:

`maxActiveAuthTokens`

(Opcional) El número máximo de tokens de autorización de dispositivos de cliente activos. Puede aumentar este número para permitir que un mayor número de dispositivos de cliente se conecten a un solo dispositivo principal sin volver a autenticarlos.

Valor predeterminado: `2500`

`cloudRequestQueueSize`

(Opcional) El número máximo de Nube de AWS solicitudes que se van a poner en cola antes de que este componente las rechace.

Valor predeterminado: `100`

`maxConcurrentCloudRequests`

(Opcional) El número máximo de solicitudes simultáneas que se enviarán a la Nube de AWS. Puede aumentar este número para mejorar el rendimiento de la autenticación en los dispositivos principales a los que se conecta un gran número de dispositivos de cliente.

Valor predeterminado: `1`

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

En el siguiente ejemplo de configuración se especifica que se permita la conexión de los dispositivos cliente cuyos nombres comiencen por: y publish/subscribe en todos los temas. `MyClientDevice`

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
```

```
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
```

```
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.1.x

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de

cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

groupNameKey

El nombre de este grupo de dispositivos. *groupNameKey*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al final del nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen por la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo,

especifique `thingName: MyTeam\\\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`: el nombre del objeto de AWS IoT del dispositivo de cliente.

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

policyNameKey

El nombre de esta política de autorización. *policyNameKey*Sustitúyala por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

statementNameKey

El nombre de esta declaración de política. *statementNameKey*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

  Esta operación admite los siguientes recursos:

  - `mqtt:clientId:`*deviceClientId*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *deviceClientId*Sustitúyalo por el ID de cliente que se va a utilizar.
- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

  - `mqtt:topic:`*mqttTopic*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *mqttTopic*Sustitúyalo por el tema que se va a utilizar.

    Este recurso no admite caracteres comodín de temas MQTT.
- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

- `mqtt:topicfilter:`*`mqttTopicFilter`*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *`mqttTopicFilter`*Sustitúyalo por el filtro de tema que desee utilizar.

  Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

  El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín * para permitir el acceso a todos los recursos. No puede usar el comodín * para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

`certificates`

(Opcional) Las opciones de configuración del certificado para este dispositivo principal. Este objeto contiene la siguiente información:

`serverCertificateValiditySeconds`

(Opcional) El periodo de tiempo (en segundos) luego del cual caduca el certificado del servidor MQTT local. Puede configurar esta opción para personalizar la frecuencia con

la que los dispositivos de cliente se desconectan y se vuelven a conectar al dispositivo principal.

Este componente rota el certificado del servidor MQTT local 24 horas antes de que caduque. El agente MQTT, como el componente agente MQTT de Moquette, genera un nuevo certificado y se reinicia. Cuando esto ocurre, todos los dispositivos de cliente conectados a este dispositivo principal se desconectan. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal tras un breve periodo.

Valor predeterminado: `604800` (7 días)

Valor mínimo: `172800` (2 días)

Valor máximo: `864000` (10 días)

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

El siguiente ejemplo de configuración especifica que los dispositivos cliente cuyos nombres comiencen por «se `MyClientDevice` conecten» y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
```

```
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
```

```
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.0.x

`deviceGroups`

Los grupos de dispositivos son grupos de dispositivos de cliente que tienen permisos para conectarse y comunicarse con un dispositivo principal. Use reglas de selección para identificar grupos de dispositivos de cliente y defina políticas de autorización de dispositivos de cliente que especifiquen los permisos para cada grupo de dispositivos.

Este objeto contiene la siguiente información:

`formatVersion`

La versión de formato de este objeto de configuración.

Puede elegir entre las siguientes opciones:

- `2021-03-05`

`definitions`

Los grupos de dispositivos de este dispositivo principal. Cada definición especifica una regla de selección para evaluar si un dispositivo de cliente es miembro del grupo. Cada definición también especifica la política de permisos que se aplicará a los dispositivos de cliente que coincidan con la regla de selección. Si un dispositivo de cliente es miembro de varios grupos de dispositivos, los permisos del dispositivo se componen de la política de permisos de cada grupo.

Este objeto contiene la siguiente información:

`groupNameKey`

El nombre de este grupo de dispositivos. *`groupNameKey`*Sustitúyalo por un nombre que ayude a identificar este grupo de dispositivos.

Este objeto contiene la siguiente información:

`selectionRule`

La consulta que especifica qué dispositivos de cliente son miembros de este grupo de dispositivos. Cuando un dispositivo de cliente se conecta, el dispositivo principal evalúa esta regla de selección para determinar si el dispositivo de cliente es miembro de este grupo de dispositivos. Si el dispositivo de cliente es miembro, el dispositivo principal usa la política de este grupo de dispositivos para autorizar las acciones del dispositivo de cliente.

Cada regla de selección incluye al menos una cláusula de regla de selección, que es una consulta de expresión única que puede coincidir con los dispositivos de cliente. Las reglas de selección utilizan la misma sintaxis de consulta que la indexación de AWS IoT flotas. Para obtener más información sobre la sintaxis de las reglas de selección, consulte la sintaxis de las consultas de indexación de flotas de AWS IoT en la Guía para desarrolladores de AWS IoT Core .

Use el comodín * para hacer coincidir varios dispositivos de cliente con una cláusula de regla de selección. Puede usar este comodín al final del nombre del objeto para hacer coincidir los dispositivos de cliente cuyos nombres comiencen por la cadena que especifique. También puede usar este comodín para hacer coincidir todos los dispositivos de cliente.

> **Note**
>
> Para seleccionar un valor que contenga dos puntos (`:`), evite los dos puntos y coloque un carácter de barra invertida (\\). En formatos como JSON, debe evitar los caracteres de barra invertida, por lo que debe escribir dos caracteres de barra invertida antes del carácter de dos puntos. Por ejemplo, especifique `thingName: MyTeam\\\\:ClientDevice1` para seleccionar un elemento cuyo nombre sea `MyTeam:ClientDevice1`.

Puede especificar el siguiente selector:

- `thingName`: el nombre del objeto de AWS IoT del dispositivo de cliente.

Example Ejemplo de regla de selección

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres son `MyClientDevice1` o `MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Ejemplo de regla de selección (usar caracteres comodín)

La siguiente regla de selección coincide con los dispositivos de cliente cuyos nombres comiencen por `MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Ejemplo de regla de selección (hacer coincidir todos los dispositivos)

La siguiente regla de selección coincide con todos los dispositivos de cliente.

```
thingName: *
```

`policyName`

La política de permisos que se aplica a los dispositivos de cliente de este grupo de dispositivos. Especifique el nombre de la política que define en el objeto `policies`.

`policies`

Las políticas de autorización de dispositivos de cliente para dispositivos de cliente que se conectan al dispositivo principal. Cada política de autorización especifica un conjunto de acciones y los recursos en los que un dispositivo de cliente puede realizar esas acciones.

Este objeto contiene la siguiente información:

`policyNameKey`

El nombre de esta política de autorización. *`policyNameKey`*Sustitúyala por un nombre que ayude a identificar esta política de autorización. Este nombre de política se usa para definir qué política se aplica a un grupo de dispositivos.

Este objeto contiene la siguiente información:

statementNameKey

El nombre de esta declaración de política. *statementNameKey*Sustitúyala por un nombre que ayude a identificar esta declaración de política.

Este objeto contiene la siguiente información:

`operations`

La lista de operaciones que permiten usar los recursos de esta política.

Puede incluir cualquiera de las siguientes operaciones:

- `mqtt:connect`: otorga permiso para conectarse al dispositivo principal. Los dispositivos de cliente deben tener este permiso para conectarse a un dispositivo principal.

  Esta operación admite los siguientes recursos:

  - `mqtt:clientId:`*deviceClientId*: restringe el acceso en función del ID de cliente que use el dispositivo de cliente para conectarse al agente MQTT del dispositivo principal. *deviceClientId*Sustitúyalo por el ID de cliente que se va a utilizar.
- `mqtt:publish`: otorga permiso para publicar mensajes MQTT en los temas.

  Esta operación admite los siguientes recursos:

  - `mqtt:topic:`*mqttTopic*: restringe el acceso en función del tema MQTT en el que un dispositivo de cliente publica un mensaje. *mqttTopic*Sustitúyalo por el tema que se va a utilizar.

    Este recurso no admite caracteres comodín de temas MQTT.
- `mqtt:subscribe`: otorga permiso para suscribirse a los filtros de temas MQTT para recibir mensajes.

  Esta operación admite los siguientes recursos:

  - `mqtt:topicfilter:`*mqttTopicFilter*: restringe el acceso en función de los temas MQTT en los que un dispositivo de cliente puede suscribirse a los mensajes. *mqttTopicFilter*Sustitúyalo por el filtro de tema que desee utilizar.

Este recurso admite los caracteres comodín de los temas MQTT + y #. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

El dispositivo de cliente puede suscribirse a los filtros de temas exactos que usted permita. Por ejemplo, si permite que el dispositivo de cliente se suscriba al recurso `mqtt:topicfilter:client/+/status`, el dispositivo de cliente puede suscribirse a `client/+/status`, pero no a `client/client1/status`.

Puede especificar el comodín * para permitir el acceso a todas las acciones.

`resources`

La lista de recursos que permiten las operaciones de esta política. Especifique los recursos que corresponden a las operaciones de esta política. Por ejemplo, puede especificar una lista de recursos de temas MQTT (`mqtt:topic:`*`mqttTopic`*) en una política que especifique la operación `mqtt:publish`.

Puede especificar el comodín * para permitir el acceso a todos los recursos. No puede usar el comodín * para hacer coincidir los identificadores de recursos parciales. Por ejemplo, puede especificar **`"resources": "*"`**, pero no puede especificar **`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Una descripción para esta declaración de política.

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política restrictiva)

El siguiente ejemplo de configuración especifica que los dispositivos cliente cuyos nombres comiencen por «se `MyClientDevice` conecten» y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
```

```
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Ejemplo: Actualización de la combinación de configuraciones (mediante una política permisiva)

El siguiente ejemplo de configuración especifica permitir la conexión de todos los dispositivos cliente y publish/subscribe en todos los temas.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.5.5	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.5.4	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.5.3	Mejoras y correcciones de errores • Soluciona un problema que impedía que los dispositivos cliente se conectaran al dispositivo principal debido a que los certificados de cliente estaban desactualizados.
2.5.2	Versión actualizada para la versión 2.14.0 de Greengrass nucleus.
2.5.1	Mejoras y correcciones de errores • Compatible con puntos de conexión de FIPS.
2.5.0	Nuevas características • Permite la sustitución de la variable `${iot:Connection.Thing.ThingName}` por recursos de políticas.

Versión	Cambios
	• Permite los recursos de políticas con caracteres comodín como `mqtt:topic:my*` .
2.4.5	Nuevas características Suma compatibilidad con prefijos comodín para seleccionar los nombres de objetos con el parámetro `selectionRule` . Mejoras y correcciones de errores Soluciona un problema que provocaba que, en algunos casos, los certificados no se actualizarán con nueva información de conectividad.
2.4.4	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.4.3	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.4.2	Nuevas características Agrega una nueva opción de configuración `startupTimeoutSeconds` .
2.4.1	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.4.0	Nuevas características • Suma compatibilidad con la autenticación del dispositivo de cliente para emitir métricas operativas que publicará el agente de telemetría. Mejoras y correcciones de errores • Soluciona un problema por el que la autenticación del dispositivo de cliente tarda más de 10 segundos en verificar la identidad de un dispositivo de cliente. • Correcciones y mejoras menores adicionales.

Versión	Cambios
2.3.2	Mejoras y correcciones de errores • Suma compatibilidad con el almacenamiento en caché de la informaci ón del nombre de host, de modo que el componente genere correctam ente los asuntos de los certificados cuando se reinicie sin conexión a Internet.
2.3.1	Mejoras y correcciones de errores • Soluciona una fuga de memoria.
2.3.0	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Nuevas características • Suma compatibilidad con la autenticación sin conexión de los dispositivos de cliente para que puedan seguir conectándose al dispositivo principal cuando este no esté conectado a Internet. • Suma compatibilidad con la autoridad de certificación proporcionada por el cliente que el dispositivo principal usa como certificado raíz para generar certificados de agente MQTT.
2.2.3	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.2.2	Mejoras y correcciones de errores • Soluciona un problema por el que el certificado del servidor MQTT local rota con más frecuencia de lo previsto en determinadas situaciones.
2.2.1	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.

Versión	Cambios
2.2.0	Nuevas características • Suma compatibilidad con componentes personalizados para llamar a las operaciones de comunicación entre procesos (IPC) a fin de autentica r y autorizar los dispositivos de cliente. Puede usar estas operaciones en un componente de agente MQTT personalizado, por ejemplo. Para obtener más información, consulte IPC: Autenticar y autorizar dispositi vos de cliente. • Agrega las opciones `maxActiveAuthTokens` , `cloudQueueSize` y `threadPoolSize` que puede configurar para ajustar el rendimiento de este componente.
2.1.0	Nuevas características • Agrega la opción `serverCertificateValiditySeconds` que puede configurar para personalizar la fecha de caducidad del certificado del servidor agente MQTT. Puede configurar el certificado del servidor para que caduque luego de entre 2 y 10 días. Mejoras y correcciones de errores • Soluciona problemas relacionados con la forma en que este component e administra las actualizaciones de restablecimiento de la configuración. • Soluciona un problema por el que el certificado del servidor MQTT local rota con más frecuencia de lo previsto en determinadas situaciones. Para aplicar esta corrección, también debe usar la versión 2.1.0 o posterior del componente agente MQTT de Moquette. • Mejora los mensajes que este componente registra cuando rota los certificados. • Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.

Versión	Cambios
2.0.3	Mejoras y correcciones de errores • Las credenciales ahora se actualizan si se rota la clave privada del dispositivo principal. • Actualizaciones para que los mensajes de registro sean más claros.
2.0.2	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.1	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.0	Versión inicial.

CloudWatch métricas

El componente de CloudWatch métricas de Amazon (`aws.greengrass.Cloudwatch`) publica métricas personalizadas de los dispositivos principales de Greengrass en Amazon. CloudWatch El componente permite a los componentes publicar CloudWatch métricas, que puede utilizar para supervisar y analizar el entorno del dispositivo principal de Greengrass. Para obtener más información, consulta Uso de CloudWatch las métricas de Amazon en la Guía del CloudWatch usuario de Amazon.

Para publicar una CloudWatch métrica con este componente, publica un mensaje en un tema al que esté suscrito este componente. De forma predeterminada, este componente se suscribe al tema `cloudwatch/metric/put` de publicación/suscripción local. Puede especificar otros temas, incluidos los temas de AWS IoT Core MQTT, al implementar este componente.

Este componente agrupa las métricas que se encuentran en el mismo espacio de nombres y las publica a CloudWatch intervalos regulares.

> **Note**
>
> Este componente proporciona una funcionalidad similar a la del conector de CloudWatch métricas de la versión 1. AWS IoT Greengrass Para obtener más información, consulte el

conector de CloudWatch métricas en la Guía para AWS IoT Greengrass desarrolladores de la versión 1.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada
- Datos de salida
- Licencias
- Archivo de registro local
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 3.2.x
- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

Para obtener información acerca de los cambios en cada versión del componente, consulte el Registro de cambios.

Tipo

v3.x

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

v2.x

Este componente es un componente de Lambda (`aws.greengrass.lambda`). El núcleo de Greengrass ejecuta la función de Lambda de este componente mediante el componente lanzador de Lambda.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

v3.x

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

v2.x

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

3.x

- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- El rol del dispositivo de Greengrass debe permitir la acción `cloudwatch:PutMetricData`, tal como se muestra en la siguiente política de IAM de ejemplo.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Para obtener más información, consulta la referencia de CloudWatch permisos de Amazon en la Guía del CloudWatch usuario de Amazon.

2.x

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- El rol del dispositivo de Greengrass debe permitir la acción `cloudwatch:PutMetricData`, tal como se muestra en la siguiente política de IAM de ejemplo.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
```

```
    }
  ]
}
```

Para obtener más información, consulta la referencia de CloudWatch permisos de Amazon en la Guía del CloudWatch usuario de Amazon.

- Para recibir los datos de salida de este componente, debe combinar la siguiente actualización de configuración para el componente del enrutador de suscripción antiguo (`aws.greengrass.LegacySubscriptionRouter`) cuando implemente este componente. Esta configuración especifica el tema en el que este componente publica las respuestas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-cloudwatch": {
        "id": "aws-greengrass-cloudwatch",
        "source": "component:aws.greengrass.Cloudwatch",
        "subject": "cloudwatch/metric/put/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-cloudwatch": {
        "id": "aws-greengrass-cloudwatch",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
  cloudwatch:version",
        "subject": "cloudwatch/metric/put/status",
        "target": "cloud"
      }
    }
  }
  ```

  - *region*Sustitúyala por la Región de AWS que utilices.
  - *version*Sustitúyala por la versión de la función Lambda que ejecuta este componente. Para encontrar la versión de la función de Lambda, debe ver la receta de la versión de

este componente que desee implementar. Abra la página de detalles de este componente en la consola de AWS IoT Greengrass y busque el par clave-valor de la función de Lambda. Este par clave-valor contiene el nombre y la versión de la función de Lambda.

> **Important**
>
> Debe actualizar la versión de la función de Lambda en el enrutador de suscripción antiguo cada vez que implemente este componente. Esto garantiza que utilice la versión correcta de la función de Lambda para la versión del componente que implemente.

Para obtener más información, consulte Crear implementaciones.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`monitoring. region.amazonaws.com`	443	Sí	Cargue CloudWatch métricas.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia.

También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

3.2.0

En la siguiente tabla se enumeran las dependencias de las versiones 3.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

3.0.0 - 3.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 3.0.0 a 3.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.4 - 2.1.9

La siguiente tabla muestra las dependencias de las versiones 2.1.4 a 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.1.4 - 2.1.8

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.4 y 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.2 - 2.1.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.2 y 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.8 - 2.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.8 a 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Rígido
Lanzador de Lambda	>=1.0.0	Rígido
Tiempos de ejecución de Lambda	>=1.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=1.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

v3.x

`PublishInterval`

(Opcional) El número máximo de segundos que se debe esperar antes de que el componente publique las métricas por lote para un determinado espacio de nombres. Para configurar el componente para que publique las métricas a medida que las reciba, es decir, sin procesamiento por lotes, especifique `0`.

El componente se publica CloudWatch después de recibir 20 métricas en el mismo espacio de nombres o después del intervalo que especifique.

> **Note**
>
> El componente no especifica el orden en el que se publican los eventos.

Este valor puede ser de 900 segundos como máximo.

Valor predeterminado: 10 segundos

`MaxMetricsToRetain`

(Opcional) El número máximo de métricas de todos los espacios de nombres que se guardarán en la memoria antes de que se reemplacen por nuevas métricas.

Este límite se aplica cuando el dispositivo principal no tiene conexión a Internet, por lo que el componente almacena las métricas en un búfer para publicarlas más adelante. Cuando el

búfer está lleno, el componente reemplaza las métricas más antiguas por otras más nuevas. Las métricas de un determinado espacio de nombres reemplazan únicamente métricas en el mismo espacio de nombres.

> **Note**
>
> Si se interrumpe el proceso de host del componente, el componente no guarda las métricas. Esto puede ocurrir durante una implementación o cuando el dispositivo principal se reinicia, por ejemplo.

Este valor debe ser de al menos 2000 métricas.

Valor predeterminado: 5000 métricas

`InputTopic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `true` para `PubSubToIoTCore`, puede usar los comodines MQTT (+ y #) en este tema.

Valor predeterminado: `cloudwatch/metric/put`

`OutputTopic`

(Opcional) El tema en el que el componente publica las respuestas de estado.

Valor predeterminado: `cloudwatch/metric/put/status`

`PubSubToIoTCore`

(Opcional) Valor de cadena que define si se deben publicar o suscribirse a los temas de MQTT de AWS IoT Core . Los valores admitidos son `true` y `false`.

Valor predeterminado: `false`

`LogLevel`

(Opcional) El nivel de registro del componente. Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARNING`

- `ERROR`
- `CRITICAL`

Valor predeterminado: `INFO`

`UseInstaller`

(Opcional) Valor booleano que define si se debe usar el script de instalación de este componente para instalar las dependencias del SDK de este componente.

Establezca este valor a `false` si desea usar un script personalizado para instalar las dependencias o si desea incluir las dependencias de tiempo de ejecución en una imagen de Linux prediseñada. Para usar este componente, debe instalar las siguientes bibliotecas, incluidas las dependencias, y ponerlas a disposición del usuario predeterminado del sistema Greengrass.

- Versión 2 de SDK para dispositivos con AWS IoT para Python
- AWS SDK para Python (Boto3)

Valor predeterminado: `true`

`PublishRegion`

(Opcional) El lugar en el que se Región de AWS van a publicar CloudWatch las métricas. Este valor anula la región predeterminada del dispositivo principal. Este parámetro solo es obligatorio para las métricas entre regiones.

`accessControl`

(Opcional) El objeto que contiene la política de autorización que permite al componente publicar y suscribirse a los temas especificados. Si especifica valores personalizados para `InputTopic` y `OutputTopic`, debe actualizar los valores de los recursos de este objeto.

Predeterminado:

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
```

```
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:pubsub:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  }
}
```

Example Ejemplo: actualización de la combinación de configuraciones

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

> **Note**
>
> La configuración predeterminada de este componente incluye los parámetros de la función de Lambda. Le recomendamos que edite solo los siguientes parámetros para configurar este componente en sus dispositivos.

`lambdaParams`

Un objeto que contiene los parámetros de la función de Lambda de este componente. Este objeto contiene la siguiente información:

`EnvironmentVariables`

Un objeto que contiene los parámetros de la función de Lambda. Este objeto contiene la siguiente información:

`PUBLISH_INTERVAL`

(Opcional) El número máximo de segundos que se debe esperar antes de que el componente publique las métricas por lote para un determinado espacio de nombres. Para configurar el componente para que publique las métricas a medida que las reciba, es decir, sin procesamiento por lotes, especifique `0`.

El componente se publica CloudWatch después de recibir 20 métricas en el mismo espacio de nombres o después del intervalo que especifiques.

> **Note**
>
> El componente no garantiza el orden de publicación de eventos.

Este valor puede ser de 900 segundos como máximo.

Valor predeterminado: 10 segundos

`MAX_METRICS_TO_RETAIN`

(Opcional) El número máximo de métricas de todos los espacios de nombres que se guardarán en la memoria antes de que se reemplacen por nuevas métricas.

Este límite se aplica cuando el dispositivo principal no tiene conexión a Internet, por lo que el componente almacena las métricas en un búfer para publicarlas más adelante. Cuando el búfer está lleno, el componente reemplaza las métricas más antiguas por otras más nuevas. Las métricas de un determinado espacio de nombres reemplazan únicamente métricas en el mismo espacio de nombres.

> **Note**
>
> Si se interrumpe el proceso de host del componente, el componente no guarda las métricas. Esto puede ocurrir durante una implementación o cuando el dispositivo principal se reinicia, por ejemplo.

Este valor debe ser de al menos 2000 métricas.

Valor predeterminado: 5000 métricas

`PUBLISH_REGION`

(Opcional) El lugar en el que se Región de AWS van a publicar CloudWatch las métricas. Este valor anula la región predeterminada del dispositivo principal. Este parámetro solo es obligatorio para las métricas entre regiones.

`containerMode`

(Opcional) El modo de almacenamiento en contenedores de este componente. Puede elegir entre las siguientes opciones:

- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.
- `GreengrassContainer`— El componente se ejecuta en un entorno de ejecución aislado dentro del AWS IoT Greengrass contenedor.

Valor predeterminado: `GreengrassContainer`

`containerParams`

(Opcional) Un objeto que contiene los parámetros de contenedor de este componente. El componente utiliza estos parámetros si se especifica `GreengrassContainer` para `containerMode`.

Este objeto contiene la siguiente información:

`memorySize`

(Opcional) La cantidad de memoria (en kilobytes) que se va a asignar al componente.

El valor predeterminado es 64 MB (65 535 KB).

`pubsubTopics`

(Opcional) Un objeto que contiene los temas a los que el componente se suscribe para recibir mensajes. Puede especificar cada tema y si el componente se suscribe a temas de MQTT AWS IoT Core o a temas locales publish/subscribe .

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`type`

(Opcional) El tipo de publish/subscribe mensajes que utiliza este componente para suscribirse a los mensajes. Puede elegir entre las siguientes opciones:

- `PUB_SUB` — Suscribirse a la mensajería de publicación/suscripción local. Si elige esta opción, el tema no podrá contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde un componente personalizado cuando especifique esta opción, consulte Publicar/suscribir mensajes locales.
- `IOT_CORE`— Suscríbase a los mensajes de AWS IoT Core MQTT. Si elige esta opción, el tema puede contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde componentes personalizados cuando especifique esta opción, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Valor predeterminado: `PUB_SUB`

`topic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `IotCore` para `type`, puede usar los comodines de MQTT (+ y #) en este tema.

Example Ejemplo: actualización de la combinación de configuraciones (modo en contenedor)

```
{
```

```
  "containerMode": "GreengrassContainer"
}
```

Example Ejemplo: actualización de la combinación de configuraciones (modo sin contenedor)

```
{
  "containerMode": "NoContainer"
}
```

Datos de entrada

Este componente acepta métricas sobre el siguiente tema y las publica en. CloudWatch De forma predeterminada, este componente se suscribe a los publish/subscribe mensajes locales. Para obtener más información sobre cómo publicar mensajes en este componente desde sus componentes personalizados, consulte Publicar/suscribir mensajes locales.

A partir de la versión 3.0.0 del componente, si lo desea, puede configurar este componente para que se suscriba a un tema de MQTT estableciendo el parámetro de configuración `PubSubToIoTCore` en `true`. Para obtener más información sobre la publicación de mensajes en un tema de MQTT en sus componentes personalizados, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Tema predeterminado: `cloudwatch/metric/put`

El mensaje acepta las siguientes propiedades. Los mensajes de entrada deben tener un formato JSON válido.

`request`

La métrica de este mensaje.

El objeto de la solicitud contiene los datos de métricas que se publicarán en CloudWatch. Los valores de métricas deben cumplir las especificaciones de la operación `PutMetricData`.

Tipo: `object` que contiene la siguiente información:

`namespace`

El espacio de nombres definido por el usuario para los datos de las métricas de esta solicitud. CloudWatch utiliza los espacios de nombres como contenedores para los puntos de datos métricos.

> **Note**
>
> No se puede especificar un espacio de nombres que comience la cadena reservada por `AWS/`.

Tipo: `string`

Patrón válido: `[^:].*`

`metricData`

Los datos de la métrica.

Tipo: `object` que contiene la siguiente información:

`metricName`

El nombre de la métrica.

Tipo: `string`

`value`

El valor de la métrica.

> **Note**
>
> CloudWatch rechaza los valores demasiado pequeños o demasiado grandes. El valor debe estar comprendido entre `8.515920e-109` y `1.174271e+108` (Base 10) o `2e-360` y `2e360` (Base 2). CloudWatch no admite valores especiales como `NaN`+`Infinity`, y-`Infinity`.

Tipo: `double`

`dimensions`

(Opcional) Las dimensiones de la métrica. Las dimensiones proporcionan información adicional acerca de la métrica y sus datos. Una métrica puede definir hasta 10 dimensiones.

Este componente incluye automáticamente una dimensión denominada `coreName`, cuyo valor es el nombre del dispositivo principal.

Tipo: `array` de objetos, cada uno de los cuales contiene la siguiente información:

`name`

(Opcional) El nombre de la dimensión.

Tipo: `string`

`value`

(Opcional) El valor de la dimensión.

Tipo: `string`

`timestamp`

(Opcional) El tiempo en el que se recibieron los datos de las métricas, expresados en segundos, en tiempo Unix.

El valor predeterminado es la hora a la que el componente recibe el mensaje.

Tipo: `double`

> **Note**
>
> Si usa las versiones entre 2.0.3 y 2.0.7 de este componente, recomendamos que recupere la marca temporal por separado para cada métrica cuando envíe varias métricas desde un solo origen. No utilice una variable para almacenar la marca de tiempo.

`unit`

(Opcional) La unidad de la métrica.

Tipo: `string`

Valores válidos: `Seconds`, `Microseconds`, `Milliseconds`, `Bytes`, `Kilobytes`, `Megabytes`, `Gigabytes`, `Terabytes`, `Bits`, `Kilobits`, `Megabits`, `Gigabits`, `Terabits`, `Percent`, `Count`, `Bytes/Second`, `Kilobytes/Second`, `Megabytes/`

`Second`, `Gigabytes/Second`, `Terabytes/Second`, `Bits/Second`, `Kilobits/Second`, `Megabits/Second`, `Gigabits/Second`, `Terabits/Second`, `Count/Second`, `None`

El valor predeterminado es `None`.

> **Note**
>
> Todas las cuotas que se aplican a la CloudWatch `PutMetricData` API se aplican a las métricas que publiques con este componente. Los siguientes límites son especialmente importantes:
>
> - Límite de 40 KB en la carga útil de la API
> - 20 métricas por solicitud de API
> - 150 transacciones por segundo (TPS) para la API de `PutMetricData`
>
> Para obtener más información, consulta las cuotas CloudWatch de servicio en la Guía del CloudWatch usuario.

Example Ejemplo de entrada

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
  }
}
```

Datos de salida

Este componente publica las respuestas como datos de salida sobre el siguiente publish/subscribe tema local de forma predeterminada. Para obtener más información sobre cómo suscribirse a los mensajes sobre este tema en sus componentes personalizados, consulte Publicar/suscribir mensajes locales.

Si lo desea, puede configurar este componente para que se publique en un tema de MQTT al establecer el parámetro de configuración `PubSubToIoTCore` en `true`. Para obtener más información sobre cómo suscribirse a mensajes sobre un tema de MQTT en sus componentes personalizados, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

> **Note**
>
> Las versiones 2.0.x de los componentes publican las respuestas como datos de salida sobre un tema de MQTT de forma predeterminada. Debe especificar el tema como parte del `subject` en la configuración del componente del enrutador de suscripciones antiguo.

Tema predeterminado: `cloudwatch/metric/put/status`

Example Ejemplo de salida: Correcto

La respuesta incluye el espacio de nombres de los datos de la métrica y el `RequestId` campo de la CloudWatch respuesta.

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

Example Ejemplo de salida: Error

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
```

```
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

> **Note**
>
> Si el componente detecta un error que se puede volver a intentar, como un error de conexión, volverá a intentar la publicación en el siguiente lote.

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- AWS SDK para Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Cloudwatch.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

v3.x

Versión	Cambios
3.2.0	Nuevas características • Agregar compatibilidad de recetas para la versión lite del núcleo de Greengrass
3.1.0	Mejoras y correcciones de errores • Suma compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red y Permita que el dispositivo principal confíe en un proxy HTTPS.
3.0.0	Esta versión del componente de CloudWatch métricas espera parámetros de configuración diferentes a los de la versión 2.x. Si usa una configuración no predeterminada para la versión 2.x y desea actualizar de la versión 2.x a la versión 3.x, debe actualizar la configuración del componente. Para obtener más información, consulte la configuración del componente de CloudWatch métricas.

Versión	Cambios
	Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Cambia el tipo de componente, de componente de Lambda a componente genérico. Este componente ya no depende del componente del enrutador de suscripciones antiguo para crear suscripciones. • Agrega un nuevo parámetro de configuración `InputTopic` para especificar el tema al que se suscribe el componente para recibir mensajes. • Agrega un nuevo parámetro de configuración `OutputTopic` para especificar el tema en el que el componente publica las respuestas de estado. • Añade un nuevo parámetro `PubSubToIoTCore` de configuración para especificar si se deben publicar o suscribirse a los temas de AWS IoT Core MQTT. • Agrega el nuevo parámetro de configuración `UseInstaller` que permite deshabilitar opcionalmente el script de instalación que instala las dependencias del componente. Mejoras y correcciones de errores Suma compatibilidad con las marcas de tiempo duplicadas en los datos de entrada.

v2.x

Versión	Cambios
2.1.8	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.

Versión	Cambios
2.1.2	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Suma compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red y Permita que el dispositivo principal confíe en un proxy HTTPS.
2.0.8	Mejoras y correcciones de errores • Suma compatibilidad con las marcas de tiempo duplicadas en los datos de entrada. • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Véase también

- Uso de CloudWatch las métricas de Amazon en la Guía del CloudWatch usuario de Amazon
- PutMetricDataen la referencia de la CloudWatch API de Amazon

AWS IoT Device Defender

El AWS IoT Device Defender componente (`aws.greengrass.DeviceDefender`) notifica a los administradores los cambios en el estado de los dispositivos principales de Greengrass. Esto puede ayudar a identificar comportamiento inusual que podrían indicar un dispositivo en riesgo. Para obtener más información, consulte AWS IoT Device Defender en la Guía para desarrolladores de AWS IoT Core .

Este componente lee las métricas del sistema en el dispositivo principal. A continuación, publica las métricas en AWS IoT Device Defender. Para obtener más información sobre cómo leer e interpretar las métricas que informa este componente, consulte las especificaciones del documento sobre las métricas del dispositivo en la Guía para desarrolladores de AWS IoT Core .

> **Note**
>
> Este componente proporciona una funcionalidad similar a la del conector Device Defender incorporado. AWS IoT Greengrass V1 Para obtener más información, consulte conector de Device Defender en la Guía para desarrolladores de AWS IoT Greengrass V1 .

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada
- Datos de salida
- Archivo de registro local
- Licencias
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 3.1.x
- 3.0.x
- 2.0.x

Para obtener información acerca de los cambios en cada versión del componente, consulte el Registro de cambios.

Tipo

v3.x

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

v2.x

Este componente es un componente de Lambda (`aws.greengrass.lambda`). El núcleo de Greengrass ejecuta la función de Lambda de este componente mediante el componente lanzador de Lambda.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

v3.x

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

v2.x

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

v3.x

- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- AWS IoT Device Defender configurado para utilizar la función de detección para supervisar las infracciones. Para obtener más información, consulte Detectar en la Guía para desarrolladores de AWS IoT Core .

v2.x

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- AWS IoT Device Defender configurado para utilizar la función de detección para supervisar las infracciones. Para obtener más información, consulte Detectar en la Guía para desarrolladores de AWS IoT Core .
- La biblioteca psutil instalada en el dispositivo principal. La versión 5.7.0 es la última versión que se verifica para que funcione con el componente.
- La biblioteca cbor instalada en el dispositivo principal. La versión 1.0.0 es la última versión que se verifica para que funcione con el componente.
- Para recibir los datos de salida de este componente, debe combinar la siguiente actualización de configuración para el componente del enrutador de suscripción antiguo (`aws.greengrass.LegacySubscriptionRouter`) cuando implemente este componente. Esta configuración especifica el tema en el que este componente publica las respuestas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
  ```

```
      "source": "component:aws.greengrass.DeviceDefender",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-device-defender": {
      "id": "aws-greengrass-device-defender",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-
defender:version",
      "subject": "$aws/things/+/defender/metrics/json",
      "target": "cloud"
    }
  }
}
```

- *region*Sustitúyala por la Región de AWS que utilices.
- *version*Sustitúyala por la versión de la función Lambda que ejecuta este componente. Para encontrar la versión de la función de Lambda, debe ver la receta de la versión de este componente que desee implementar. Abra la página de detalles de este componente en la consola de AWS IoT Greengrass y busque el par clave-valor de la función de Lambda. Este par clave-valor contiene el nombre y la versión de la función de Lambda.

> **Important**
>
> Debe actualizar la versión de la función de Lambda en el enrutador de suscripción antiguo cada vez que implemente este componente. Esto garantiza que utilice la versión correcta de la función de Lambda para la versión del componente que implemente.

Para obtener más información, consulte Crear implementaciones.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

3.1.1

La siguiente tabla muestra las dependencias de la versión 3.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

3.0.0 - 3.0.2

La siguiente tabla muestra las dependencias de las versiones 3.0.0 a 3.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.12 - 2.0.17

En la siguiente tabla se enumeran las dependencias de las versiones 2.0.12 a 2.0.17 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.12 - 2.0.16

La siguiente tabla muestra las dependencias de la versión 2.0.16 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.10 - 2.0.11

La siguiente tabla muestra las dependencias de las versiones 2.0.10 y 2.0.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.9

En la siguiente tabla, se muestran las dependencias de la versión 2.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.8

En la siguiente tabla, se muestran las dependencias de la versión 2.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Rígido
Lanzador de Lambda	>=1.0.0	Rígido
Tiempos de ejecución de Lambda	>=1.0.0	Flexible
Servicio de intercambio de token	>=1.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

v3.x

`PublishRetryCount`

El número de veces que se volverá a intentar la publicación. Esta característica está disponible en la versión 3.1.1.

El mínimo es 0.

El máximo es 72.

Valor predeterminado: 5

`SampleIntervalSeconds`

(Opcional) La cantidad de tiempo en segundos entre cada ciclo en el que el componente recopila las métricas y las informa.

El valor mínimo es de 300 segundos (5 minutos).

Predeterminado: 300 segundos

`UseInstaller`

(Opcional) Valor booleano que define si se debe utilizar el script de instalación de este componente para instalar las dependencias de este componente.

Establezca este valor a `false` si desea usar un script personalizado para instalar las dependencias o si desea incluir las dependencias de tiempo de ejecución en una imagen de Linux prediseñada. Para usar este componente, debe instalar las siguientes bibliotecas, incluidas las dependencias, y ponerlas a disposición del usuario predeterminado del sistema Greengrass.

- Versión 2 de SDK para dispositivos con AWS IoT para Python
- Biblioteca cbor. La versión 1.0.0 es la última versión que se verifica para que funcione con el componente.
- Biblioteca psutil. La versión 5.7.0 es la última versión que se verifica para que funcione con el componente.

> **Note**
>
> Si utiliza la versión 3.0.0 o 3.0.1 de este componente en los dispositivos principales que configura para usar un proxy HTTPS, debe establecer este valor en `false`. El script de instalación no admite la operación detrás de un proxy HTTPS en estas versiones de este componente.

Valor predeterminado: `true`

v2.x

> **Note**
>
> La configuración predeterminada de este componente incluye los parámetros de la función de Lambda. Le recomendamos que edite solo los siguientes parámetros para configurar este componente en sus dispositivos.

`lambdaParams`

Un objeto que contiene los parámetros de la función de Lambda de este componente. Este objeto contiene la siguiente información:

`EnvironmentVariables`

Un objeto que contiene los parámetros de la función de Lambda. Este objeto contiene la siguiente información:

`PROCFS_PATH`

(Opcional) La ruta a la carpeta `/proc`.

- Para ejecutar este componente en un contenedor, utilice el valor predeterminado, `/host-proc`. El componente se ejecuta en un contenedor de forma predeterminada.
- Para ejecutar este componente en modo sin contenedor, especifique `/proc` para este parámetro.

Predeterminado: `/host-proc`. Esta es la ruta por defecto en la que este componente monta la carpeta `/proc` en el contenedor.

> **Note**
>
> Este componente tiene acceso de solo lectura a esta carpeta.

`SAMPLE_INTERVAL_SECONDS`

(Opcional) La cantidad de tiempo en segundos entre cada ciclo en el que el componente recopila las métricas y las informa.

El valor mínimo es de 300 segundos (5 minutos).

Predeterminado: 300 segundos

`containerMode`

(Opcional) El modo de almacenamiento en contenedores de este componente. Puede elegir entre las siguientes opciones:

- `GreengrassContainer`— El componente se ejecuta en un entorno de ejecución aislado dentro del contenedor. AWS IoT Greengrass
- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.

Si especifica esta opción, debe especificar `/proc` para el parámetro de la variable de entorno `PROCFS_PATH`.

Valor predeterminado: `GreengrassContainer`

`containerParams`

(Opcional) Un objeto que contiene los parámetros de contenedor de este componente. El componente utiliza estos parámetros si se especifica `GreengrassContainer` para `containerMode`.

Este objeto contiene la siguiente información:

`memorySize`

(Opcional) La cantidad de memoria (en kilobytes) que se va a asignar al componente.

El valor predeterminado es 50 000 KB.

`pubsubTopics`

(Opcional) Un objeto que contiene los temas a los que el componente se suscribe para recibir mensajes. Puede especificar cada tema y si el componente se suscribe a temas de MQTT AWS IoT Core o a temas locales publish/subscribe .

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`type`

(Opcional) El tipo de publish/subscribe mensajes que utiliza este componente para suscribirse a los mensajes. Puede elegir entre las siguientes opciones:

- `PUB_SUB` — Suscribirse a la mensajería de publicación/suscripción local. Si elige esta opción, el tema no podrá contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde un componente personalizado cuando especifique esta opción, consulte Publicar/suscribir mensajes locales.
- `IOT_CORE`— Suscríbase a los mensajes de AWS IoT Core MQTT. Si elige esta opción, el tema puede contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde componentes personalizados cuando especifique esta opción, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Valor predeterminado: `PUB_SUB`

`topic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `IotCore` para `type`, puede usar los comodines de MQTT (+ y #) en este tema.

Example Ejemplo: actualización de la combinación de configuraciones (modo en contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Ejemplo: actualización de la combinación de configuraciones (modo sin contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

Datos de entrada

Este componente no acepta mensajes como datos de entrada.

Datos de salida

Este componente publica las métricas de seguridad en el siguiente tema reservado para AWS IoT Device Defender. Este componente se *`coreDeviceName`* sustituye por el nombre del dispositivo principal cuando publica las métricas.

Tema (AWS IoT Core MQTT): `$aws/things/`*`coreDeviceName`*`/defender/metrics/json`

Example Ejemplo de resultado


```
{
  "header": {
    "report_id": 1529963534,
    "version": "1.0"
  },
  "metrics": {
    "listening_tcp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 24800
        },
        {
          "interface": "eth0",
          "port": 22
        },
        {
          "interface": "eth0",
          "port": 53
        }
      ],
      "total": 3
    },
    "listening_udp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 5353
        },
        {
          "interface": "eth0",
          "port": 67
        }
      ],
      "total": 2
    },
    "network_stats": {
      "bytes_in": 1157864729406,
      "bytes_out": 1170821865,
      "packets_in": 693092175031,
      "packets_out": 738917180
    },
```

```
    "tcp_connections": {
      "established_connections":{
        "connections": [
          {
            "local_interface": "eth0",
            "local_port": 80,
            "remote_addr": "192.168.0.1:8000"
          },
          {
            "local_interface": "eth0",
            "local_port": 80,
            "remote_addr": "192.168.0.1:8000"
          }
        ],
        "total": 2
      }
    }
  }
}
```

Para obtener más información sobre las métricas que informa este componente, consulte las especificaciones del documento sobre las métricas de los dispositivos en la Guía para desarrolladores de AWS IoT Core .

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -
Wait
```

Licencias

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

v3.x

Versión	Cambios
3.1.1	Mejoras y correcciones de errores • Agrega reintentos para la conexión del cliente cuando la conexión no se recupera después de una interrupción de la red. • Agrega un reintento configurable para publicar métricas.
3.1.0	Mejoras y correcciones de errores • Suma compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red y Permita que el dispositivo principal confíe en un proxy HTTPS.
3.0.1	Soluciona un problema relacionado con la forma en que el componente calcula los valores delta de las métricas.

Versión	Cambios
3.0.0	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Versión inicial.

v2.x

Versión	Cambios
2.0.17	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.0.16	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.0.11	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.0.10	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.0.9	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.

Versión	Cambios
2.0.5	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Spooler de disco

El componente disk spooler (`aws.greengrass.DiskSpooler`) ofrece una opción de almacenamiento persistente para los mensajes enviados desde los dispositivos principales de Greengrass a. AWS IoT Core Este componente almacenará estos mensajes salientes en el disco.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- De uso
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 1.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- `storageType` debería configurarse a `Disk` para usar este componente. Puede configurarlo en la configuración del núcleo de Greengrass.
- `maxSizeInBytes` no debe configurarse para que supere el espacio disponible en el dispositivo. Puede configurarlo en la configuración del núcleo de Greengrass.
- Se admite la ejecución del componente del spooler de disco en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

1.0.7

En la siguiente tabla, se muestran las dependencias de la versión 1.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 <2.17.0	Rígido

1.0.

En la siguiente tabla, se muestran las dependencias de la versión 1.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 =2.11.0 <2.16.0	Rígido

1.0.5

En la siguiente tabla, se muestran las dependencias de la versión 1.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 =2.11.0 <2.15.0	Rígido

1.0.4

En la siguiente tabla, se muestran las dependencias de la versión 1.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 <2.14.0	Rígido

1.0.1 – 1.0.3

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.1 a 1.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 <2.13.0	Rígido

1.0.0

En la siguiente tabla, se muestran las dependencias de la versión 1.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 <2.12.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

De uso

Para utilizar el componente del spooler de disco, `aws.greengrass.DiskSpooler` debe estar implementado.

Para configurar y usar este componente, debe configurar el `pluginName` en `aws.greengrass.DiskSpooler`.

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o por la ruta a la *C:\greengrass\v2* carpeta raíz. AWS IoT Greengrass

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.0.7	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
1.0.6	Versión actualizada para el lanzamiento de la versión 2.15.0 de Greengrass nucleus.
1.0.5	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
1.0.4	Mejoras y correcciones de errores Correcciones de errores generales.
1.0.3	Mejoras y correcciones de errores Mejora el rendimiento al reutilizar las conexiones de la base de datos.
1.0.2	Mejoras y correcciones de errores Soluciona un problema por el que el campo de formato de mensaje MQTT no se conserva en algunos casos.

Versión	Cambios
1.0.1	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
1.0.0	Versión inicial.

Administrador de aplicaciones de Docker

El componente gestor de aplicaciones de Docker (`aws.greengrass.DockerApplicationManager`) AWS IoT Greengrass permite descargar imágenes de Docker de registros de imágenes públicos y registros privados alojados en Amazon Elastic Container Registry (Amazon ECR). También permite gestionar las credenciales automáticamente AWS IoT Greengrass para descargar imágenes de forma segura desde repositorios privados en Amazon ECR.

Cuando desarrolle un componente personalizado que ejecute un contenedor de Docker, incluya el administrador de aplicaciones de Docker como dependencia para descargar las imágenes de Docker especificadas como artefactos en su componente. Para obtener más información, consulte Ejecución de un contenedor de Docker.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- Versión 1.9.1 o posterior de Docker Engine instalada en el dispositivo principal de Greengrass. Se ha comprobado que la versión 20.10 es la última versión que funciona con el AWS IoT Greengrass software Core. Debe instalar Docker directamente en el dispositivo principal antes de implementar componentes que ejecuten contenedores de Docker.
- El daemon de Docker se inició y se ejecutó en el dispositivo principal antes de implementar este componente.
- Imágenes de Docker almacenadas en uno de los siguientes orígenes de imágenes compatibles:
  - Repositorios de imágenes públicos y privados en Amazon Elastic Container Registry (Amazon ECR)
  - Repositorio público de Docker Hub
  - Registro público de confianza de Docker
- Las imágenes de Docker se incluyen como artefactos en sus componentes de contenedor de Docker personalizados. Use los siguientes formatos de URI para especificar sus imágenes de Docker:
  - Imagen privada de Amazon ECR: `docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`

- Imagen pública de Amazon ECR: `docker:public.ecr.aws/`*`repository`*`/`*`image`*`[:tag|@digest]`
- Imagen pública de Docker Hub: `docker:`*`name`*`[:tag|@digest]`

Para obtener más información, consulte Ejecución de un contenedor de Docker.

> **Note**
>
> Si no especifica la etiqueta de la imagen o el resumen de la imagen en el URI del artefacto de una imagen, el administrador de aplicaciones de Docker extrae la última versión disponible de esa imagen al implementar el componente del contenedor de Docker personalizado. Para asegurarse de que todos sus dispositivos principales ejecuten la misma versión de una imagen, recomendamos que incluya la etiqueta o el resumen de la imagen en el URI del artefacto.

- El usuario del sistema que ejecute un componente contenedor de Docker debe tener permisos de raíz o administrador, o bien debe configurar Docker para que se ejecute como un usuario no de raíz o no administrador.
  - En los dispositivos Linux, debe agregar un usuario al grupo de `docker` para llamar comandos `docker` sin `sudo`.
  - En los dispositivos Windows, puede agregar un usuario al grupo de `docker-users` para llamar comandos `docker` sin privilegios de administrador.

  Linux or Unix

  Para agregar `ggc_user`, o el usuario no raíz que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker`, ejecute el siguiente comando.

  ```
  sudo usermod -aG docker ggc_user
  ```

  Para obtener más información, consulte Administrar Docker como un usuario no raíz.

  Windows Command Prompt (CMD)

  Para agregar `ggc_user`, o el usuario que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker-users`, ejecute el siguiente comando como un administrador.

  ```
  net localgroup docker-users ggc_user /add
  ```

Windows PowerShell

Para agregar `ggc_user`, o el usuario que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker-users`, ejecute el siguiente comando como un administrador.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Si configura el software AWS IoT Greengrass Core para usar un proxy de red, debe configurar Docker para que use el mismo servidor proxy.
- Si sus imágenes de Docker están almacenadas en un registro privado de Amazon ECR, debe incluir el componente del servicio de intercambio de token como una dependencia en el componente del contenedor de Docker. Además, el rol de dispositivo de Greengrass debe permitir las acciones `ecr:GetAuthorizationToken`, `ecr:BatchGetImage` y `ecr:GetDownloadUrlForLayer` como se muestra en el siguiente ejemplo de política de IAM.

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": [
          "ecr:GetAuthorizationToken",
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
        ],
        "Resource": [
          "*"
        ],
        "Effect": "Allow"
      }
    ]
  }
  ```

- Se admite la ejecución del componente del administrador de aplicaciones de Docker en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.

- El componente del administrador de aplicaciones de Docker debe tener conectividad para descargar imágenes. Por ejemplo, si usa ECR, debe tener conectividad con los siguientes puntos de conexión.
  - `*.dkr.ecr.region.amazonaws.com` (punto de conexión de VPC `com.amazonaws.region.ecr.dkr`)
  - `api.ecr.region.amazonaws.com` (punto de conexión de VPC `com.amazonaws.region.ecr.api`)

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`ecr.region.amazonaws.com`	443	No	Obligatorio si descarga imágenes de Docker de Amazon ECR.

`hub.docker.com`
 	
`registry.hub.docker.com/v1`
	443	No	Obligatorio si descarga imágenes de Docker desde Docker Hub.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.0.15

En la siguiente tabla, se muestran las dependencias de la versión 2.0.15 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.17.0	Flexible

2.0.14

La siguiente tabla muestra las dependencias de la versión 2.0.14 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.16.0	Flexible

2.0.13

La siguiente tabla muestra las dependencias de la versión 2.0.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.15.0	Flexible

2.0.12

En la siguiente tabla, se muestran las dependencias de la versión 2.0.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.14.0	Flexible

2.0.11

En la siguiente tabla, se muestran las dependencias de la versión 2.0.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.13.0	Flexible

2.0.10

En la siguiente tabla, se muestran las dependencias de la versión 2.0.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.12.0	Flexible

2.0.9

En la siguiente tabla, se muestran las dependencias de la versión 2.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.11.0	Flexible

2.0.8

En la siguiente tabla, se muestran las dependencias de la versión 2.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.10.0	Flexible

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.9.0	Flexible

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.8.0	Flexible

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.7.0	Flexible

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.6.0	Flexible

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.5.0	Flexible

2.0.2

En la siguiente tabla, se muestran las dependencias de la versión 2.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.4.0	Flexible

2.0.1

En la siguiente tabla, se muestran las dependencias de la versión 2.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.3.0	Flexible

2.0.0

En la siguiente tabla, se muestran las dependencias de la versión 2.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.2.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o por la ruta a *C:\greengrass\v2* la carpeta raíz. AWS IoT Greengrass

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.0.15	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.0.14	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.0.13	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.

Versión	Cambios
2.0.12	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.0.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.0.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.0.9	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.0	Versión inicial.

Véase también

- Ejecución de un contenedor de Docker

Conector periférico para Kinesis Video Streams

El conector periférico para el componente Kinesis Video Streams (`aws.iot.EdgeConnectorForKVS`) lee las transmisiones de video de las cámaras locales y las publica en Kinesis Video Streams. Puede configurar este componente para leer las transmisiones de video de las cámaras de protocolo de Internet (IP) mediante el protocolo de transmisión en tiempo real (RTSP). A continuación, puede configurar paneles en los servidores de Amazon Managed Grafana o Grafana locales para supervisar los flujos de video e interactuar con ellos.

Puede integrar este componente AWS IoT TwinMaker para mostrar y controlar las transmisiones de vídeo en los paneles de Grafana. AWS IoT TwinMaker es un AWS servicio que le permite crear gemelos digitales operativos de sistemas físicos. Puede utilizarlo AWS IoT TwinMaker para visualizar los datos de los sensores, las cámaras y las aplicaciones empresariales con el fin de realizar un seguimiento de sus fábricas, edificios o plantas industriales físicas. También puede utilizar estos datos para supervisar las operaciones, diagnosticar errores y repararlos. Para obtener más información, consulte ¿Qué es AWS IoT TwinMaker? en la Guía AWS IoT TwinMaker del usuario.

Este componente almacena su configuración en AWS IoT SiteWise, que es un AWS servicio que modela y almacena datos industriales. En AWS IoT SiteWise, los activos representan objetos como dispositivos, equipos o grupos de otros objetos. Para configurar y usar este componente, debe crear un AWS IoT SiteWise activo para cada dispositivo principal de Greengrass y para cada cámara IP conectada a cada dispositivo principal. Cada activo tiene propiedades que se configuran para controlar características, como la transmisión en directo, la carga bajo demanda y el almacenamiento en caché local. Para especificar la URL de cada cámara, debe crear una entrada secreta en AWS Secrets Manager que contenga la URL de la cámara. Si la cámara requiere autenticación, también debe especificar un nombre de usuario y una contraseña en la URL. A continuación, especifique ese secreto en una propiedad de activo de la cámara de IP.

Este componente carga la transmisión de video de cada cámara a una transmisión de video de Kinesis. Debe especificar el nombre de la transmisión de vídeo de Kinesis de destino en la configuración de AWS IoT SiteWise activos de cada cámara. Si la transmisión de video de Kinesis no existe, este componente la crea por usted.

AWS IoT TwinMaker proporciona un script que puede ejecutar para crear estos AWS IoT SiteWise activos y los secretos de Secrets Manager. Para obtener más información sobre cómo crear estos recursos y cómo instalar, configurar y usar este componente, consulte la integración de AWS IoT TwinMaker vídeo en la Guía del AWS IoT TwinMaker usuario.

> **Note**
>
> El conector periférico para Kinesis Video Streams solo está disponible en las siguientes Regiones de AWS:
>
> - Este de EE. UU. (Norte de Virginia)
> - Oeste de EE. UU. (Oregón)
> - Europa (Fráncfort)
> - Europa (Irlanda)
> - Asia-Pacífico (Singapur)
> - Asia-Pacífico (Tokio)
> - Asia-Pacífico (Seúl)
> - Asia-Pacífico (Sídney)
> - Asia-Pacífico (Mumbai)
> - China (Pekín)

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Licencias
- De uso
- Archivo de registro local
- Registros de cambios

- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 1.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- Puede implementar este componente solo en dispositivos de un núcleo principal, ya que la configuración del componente debe ser única para cada dispositivo principal. No puede implementar este componente en grupos de dispositivos principales.
- GStreamer1.18.4 o una versión posterior instalada en el dispositivo principal. Para obtener más información, consulte Instalación. GStreamer

  En un dispositivo con`apt`, puede ejecutar los siguientes comandos para realizar la instalación GStreamer.

```
sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
 gstreamer1.0-plugins-base-apps
sudo apt install -y gstreamer1.0-libav
sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
plugins-ugly gstreamer1.0-tools
```

- Un AWS IoT SiteWise activo para cada dispositivo principal. Este AWS IoT SiteWise activo representa el dispositivo principal. Para obtener más información sobre cómo crear este recurso, consulte la integración de AWS IoT TwinMaker vídeo en la Guía del AWS IoT TwinMaker usuario.
- Un AWS IoT SiteWise recurso para cada cámara IP que se conecte a cada dispositivo principal. Estos activos AWS IoT SiteWise representan las cámaras que transmiten video a cada dispositivo principal. El activo de cada cámara debe estar asociado al activo del dispositivo principal que se conecta a la cámara. Los activos de cámara tienen propiedades que puede configurar para especificar un flujo de video de Kinesis, un secreto de autenticación y parámetros de transmisión de video. Para obtener más información sobre cómo crear y configurar los activos de la cámara, consulte la integración de AWS IoT TwinMaker vídeo en la Guía del AWS IoT TwinMaker usuario.
- Un AWS Secrets Manager secreto para cada cámara IP. Este secreto debe definir un par clave-valor, donde la clave es `RTSPStreamUrl` y el valor es la URL de la cámara. Si la cámara requiere autenticación, incluya el nombre de usuario y la contraseña en esta URL. Puede usar una cadena para crear un secreto cuando cree los recursos que requiere este componente. Para obtener más información, consulte la integración de AWS IoT TwinMaker vídeo en la Guía del AWS IoT TwinMaker usuario.

  También puede utilizar la consola del administrador de secretos y la API para crear secretos adicionales. Para obtener más información, consulte Cómo crear un secreto en la Guía del usuario de AWS Secrets Manager .
- La función de intercambio de tokens de Greengrass debe permitir las siguientes acciones y las de Kinesis Video Streams AWS Secrets Manager AWS IoT SiteWise, como se muestra en el siguiente ejemplo de política de IAM.

> **Note**
>
> Este ejemplo de política permite al dispositivo obtener el valor de los secretos con nombres **`IPCamera1Url`** y **`IPCamera2Url`**. Cuando configura cada cámara de IP, se especifica un secreto que contiene la URL de esa cámara. Si la cámara requiere autenticación, también debe especificar un nombre de usuario y una contraseña en la URL. El rol de intercambio de token del dispositivo principal debe permitir el acceso al secreto de cada cámara de IP a la que se conecte.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
      "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera1Url",
  "arn:aws:secretsmanager:us-east-1:123456789012:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
        "iotsitewise:ListAssociatedAssets",
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:PutMedia",
        "kinesisvideo:TagStream"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

> **Note**
>
> Si utiliza una AWS Key Management Service clave gestionada por el cliente para cifrar los secretos, la función de dispositivo también debe permitir la acción. `kms:Decrypt`

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`kinesisvideo.` *`region`*`.amazonaws.com`	443	Sí	Cargue datos a Kinesis Video Streams.
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Sí	Publique los metadatos del flujo de video en AWS IoT SiteWise.
`secretsmanager.` *`region`*`.amazonaws.com`	443	Sí	Descargue los secretos de la URL de la cámara al

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			dispositivo principal.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.0 a 1.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=2.0.3	Rígido
Administrador de flujos	>=2.0.9	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`SiteWiseAssetIdForHub`

El ID del AWS IoT SiteWise activo que representa este dispositivo principal. Para obtener más información sobre cómo crear este recurso y usarlo para interactuar con este componente, consulte la integración de AWS IoT TwinMaker vídeo en la Guía del AWS IoT TwinMaker usuario.

Example Ejemplo: actualización de la combinación de configuraciones

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- Quartz Job Scheduler/Apache License 2.0
- Vinculaciones de Java para la GStreamer licencia pública general reducida 1.x/GNU v3.0

De uso

Para configurar este componente e interactuar con él, puede configurar las propiedades de AWS IoT SiteWise los activos que representan el dispositivo principal y las cámaras IP a las que se conecta. También puede visualizar e interactuar con las transmisiones de vídeo en los paneles de Grafana a través de. AWS IoT TwinMaker Para obtener más información, consulte la integración de AWS IoT TwinMaker vídeo en la Guía del AWS IoT TwinMaker usuario.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyala por la ruta a la carpeta AWS IoT Greengrass raíz.

```
sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.0.5	Corrección de errores y mejoras generales.
1.0.4	Mejoras y correcciones de errores • Soluciona un problema que provocaba que se detuviera la carga en directo.
1.0.3	Corrección de errores y mejoras generales.
1.0.1	Corrección de errores y mejoras generales.
1.0.0	Versión inicial.

Véase también

- ¿Qué es AWS IoT TwinMaker? en la Guía AWS IoT TwinMaker del usuario
- AWS IoT TwinMaker integración de vídeo en la Guía AWS IoT TwinMaker del usuario
- ¿Qué es AWS IoT SiteWise? en la Guía AWS IoT SiteWise del usuario
- Actualización de los valores de los atributos en la Guía del usuario de AWS IoT SiteWise
- ¿Qué es AWS Secrets Manager? en la Guía del usuario de AWS Secrets Manager
- Creación y administración de secretos en la Guía del usuario de AWS Secrets Manager

CLI de Greengrass

El componente CLI de Greengrass (`aws.greengrass.Cli`) proporciona una interfaz de línea de comandos local que puede usar en los dispositivos principales para desarrollar y depurar componentes de forma local. La CLI de Greengrass le permite crear implementaciones locales y reiniciar componentes en el dispositivo principal, por ejemplo.

Puede instalar este componente al instalar el software principal. AWS IoT Greengrass Para obtener más información, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

> **Important**
>
> Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Una vez instalado este componente, ejecute el siguiente comando para ver la documentación de ayuda. Cuando se instala este componente, se agrega un enlace simbólico a `greengrass-cli` en la carpeta */greengrass/v2*`/bin`. Puede ejecutar la CLI de Greengrass desde esta ruta o agregarla a la variable de entorno `PATH` para que `greengrass-cli` se ejecute sin la ruta absoluta.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

El siguiente comando reinicia un componente denominado `com.example.HelloWorld`, por ejemplo.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Para obtener más información, consulte Interfaz de la línea de comandos de Greengrass.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.16.x
- 2.15.x
- 2.14.x
- 2.13.x
- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- Debe estar autorizado a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Realice una de las siguientes acciones para usar la CLI de Greengrass:
  - Utilice el usuario del sistema que ejecuta el software AWS IoT Greengrass Core.
  - Utilice un usuario con permisos raíz o administrativos. En los dispositivos principales de Linux, puede utilizar `sudo` para obtener permisos de raíz.
  - Utilice un usuario del sistema que esté en un grupo que especifique en los parámetros de configuración `AuthorizedPosixGroups` o `AuthorizedWindowsGroups` al implementar el componente. Para obtener más información, consulte Configuración del componente de la CLI de Greengrass.
- Se admite la ejecución del componente CLI de Greengrass en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas

sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.16.0

La siguiente tabla muestra las dependencias de la versión 2.16.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.12.0 <2.17.0	Flexible

2.15.1

La siguiente tabla muestra las dependencias de la versión 2.15.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.12.0 <2.16.0	Flexible

2.15.0

La siguiente tabla muestra las dependencias de la versión 2.15.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.12.0 <2.16.0	Flexible

2.14.0 – 2.14.3

La siguiente tabla muestra las dependencias de las versiones 2.14.0 y 2.14.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.12.0 <2.15.0	Flexible

2.13.0

La siguiente tabla muestra las dependencias de la versión 2.13.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.12.0 <2.14.0	Flexible

2.12.0 – 2.12.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.12.0 a 2.12.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.12.0 <2.13.0	Flexible

2.11.0 – 2.11.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.11.0 a 2.11.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.11.0 <2.12.0	Flexible

2.10.0 – 2.10.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.10.0 a 2.10.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.11.0	Flexible

2.9.0 – 2.9.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.9.0 a 2.9.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.10.0	Flexible

2.8.0 – 2.8.1

En la siguiente tabla, se muestran las dependencias de las versiones 2.8.0 y 2.8.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.9.0	Flexible

2.7.0

En la siguiente tabla, se muestran las dependencias de la versión 2.7.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.8.0	Flexible

2.6.0

En la siguiente tabla, se muestran las dependencias de la versión 2.6.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.7.0	Flexible

2.5.0 – 2.5.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.5.0 a 2.5.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.6.0	Flexible

2.4.0

En la siguiente tabla, se muestran las dependencias de la versión 2.4.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.5.0	Flexible

2.3.0

En la siguiente tabla, se muestran las dependencias de la versión 2.3.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.4.0	Flexible

2.2.0

En la siguiente tabla, se muestran las dependencias de la versión 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.3.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.2.0	Flexible

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.1.0	Flexible

> **Note**
>
> La versión mínima compatible del núcleo de Greengrass corresponde a la versión de revisión del componente CLI de Greengrass.

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.5.x - 2.14.x

`AuthorizedPosixGroups`

(Opcional) Una cadena que contiene una lista de grupos de sistema separados por comas. Usted autoriza a estos grupos de sistemas a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Puede especificar los nombres de los grupos

o grupos IDs. Por ejemplo, `group1,1002,group3` autoriza a tres grupos de sistemas (`group1`, `1002` y `group3`) a utilizar la CLI de Greengrass.

Si no especifica ningún grupo para autorizarlo, puede usar la CLI de Greengrass como usuario root (`sudo`) o como usuario del sistema que ejecuta el software AWS IoT Greengrass Core.

`AuthorizedWindowsGroups`

(Opcional) Una cadena que contiene una lista de grupos de sistema separados por comas. Usted autoriza a estos grupos de sistemas a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Puede especificar los nombres de los grupos o grupos IDs. Por ejemplo, `group1,1002,group3` autoriza a tres grupos de sistemas (`group1`, `1002` y `group3`) a utilizar la CLI de Greengrass.

Si no especifica ningún grupo para autorizarlo, puede usar la CLI de Greengrass como administrador o como usuario del sistema que ejecuta el software AWS IoT Greengrass principal.

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo de configuración, se especifica que se debe autorizar a tres grupos del sistema POSIX (`group1`, `1002` y `group3`) y dos grupos de usuarios de Windows (`Device Operators` y `QA Engineers`) a utilizar la CLI de Greengrass.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

`AuthorizedPosixGroups`

(Opcional) Una cadena que contiene una lista de grupos de sistema separados por comas. Usted autoriza a estos grupos de sistemas a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Puede especificar los nombres de los grupos o grupos IDs. Por ejemplo, `group1,1002,group3` autoriza a tres grupos de sistemas (`group1`, `1002` y `group3`) a utilizar la CLI de Greengrass.

Si no especifica ningún grupo para autorizarlo, puede usar la CLI de Greengrass como usuario root (`sudo`) o como usuario del sistema que ejecuta el software AWS IoT Greengrass Core.

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo de configuración, se especifica que se autorice a tres grupos de sistemas (`group1`, `1002` y `group3`) a utilizar la CLI de Greengrass.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.16.1	Versión actualizada para la versión 2.16.1 de Greengrass Nucleus.
2.16.0	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.15.1	Versión actualizada para la versión 2.15.1 de Greengrass Nucleus.
2.15.0	Mejoras y correcciones de errores • Agrega una solución para que, cuando la configuración de autenticación cambie, el componente no se vuelva a instalar, sino que se reinicie. • Corrección de errores y mejoras generales.
2.14.3	Versión actualizada para la versión 2.14.3 de Greengrass Nucleus.
2.14.2	Versión actualizada para la versión 2.14.2 de Greengrass Nucleus.
2.14.1	Versión actualizada para la versión 2.14.1 de Greengrass Nucleus.
2.14.0	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Valide el parámetro de destino de implementación en el comando de CLI.
2.13.0	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.12.6	Versión actualizada de la versión 2.12.6 del núcleo de Greengrass.
2.12.5	Versión actualizada para el lanzamiento de la versión 2.12.5 del núcleo de Greengrass.
2.12.4	Versión actualizada para el lanzamiento de la versión 2.12.4 del núcleo de Greengrass.

Versión	Cambios
2.12.3	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Versión actualizada para el lanzamiento de la versión 2.12.3 del núcleo de Greengrass.
2.12.2	Versión actualizada para el lanzamiento de la versión 2.12.2 del núcleo de Greengrass.
2.12.1	Versión actualizada para el lanzamiento de la versión 2.12.1 del núcleo de Greengrass.
2.12.0	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.11.3	Versión actualizada para el lanzamiento de la versión 2.11.3 del núcleo de Greengrass.
2.11.2	Versión actualizada para el lanzamiento de la versión 2.11.2 del núcleo de Greengrass.
2.11.1	Versión actualizada para el lanzamiento de la versión 2.11.1 del núcleo de Greengrass.
2.11.0	Nuevas características • Le permite cancelar una implementación local. • Le permite configurar una política de gestión de errores para una implementación local. • Mejora los informes detallados del estado de la implementación.
2.10.3	Versión actualizada para el lanzamiento de la versión 2.10.3 del núcleo de Greengrass.

Versión	Cambios
2.10.2	Versión actualizada para el lanzamiento de la versión 2.10.2 del núcleo de Greengrass.
2.10.1	Versión actualizada para el lanzamiento de la versión 2.10.1 del núcleo de Greengrass.
2.10.0	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.9.6	Versión actualizada para el lanzamiento de la versión 2.9.6 del núcleo de Greengrass.
2.9.5	Versión actualizada para el lanzamiento de la versión 2.9.5 del núcleo de Greengrass.
2.9.4	Versión actualizada para el lanzamiento de la versión 2.9.4 del núcleo de Greengrass.
2.9.3	Versión actualizada para el lanzamiento de la versión 2.9.3 del núcleo de Greengrass.
2.9.2	Versión actualizada para el lanzamiento de la versión 2.9.2 del núcleo de Greengrass.
2.9.1	Versión actualizada para el lanzamiento de la versión 2.9.1 del núcleo de Greengrass.
2.9.0	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.8.1	Versión actualizada para el lanzamiento de la versión 2.8.1 del núcleo de Greengrass.
2.8.0	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.7.0	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.

Versión	Cambios
2.6.0	Nuevas características • Agrega compatibilidad con componentes personalizados para llamar a las operaciones de comunicación entre procesos (IPC) que utiliza la CLI de Greengrass. Puede usar estas operaciones de IPC para administr ar las implementaciones locales, ver los detalles de los component es y generar una contraseña que podrá usar para iniciar sesión en la consola de depuración local. Para obtener más información, consulte IPC: administrar las implementaciones y los componentes locales. Mejoras y correcciones de errores • Correcciones y mejoras menores adicionales.
2.5.6	Versión actualizada para el lanzamiento de la versión 2.5.6 del núcleo de Greengrass.
2.5.5	Versión actualizada para el lanzamiento de la versión 2.5.5 del núcleo de Greengrass.
2.5.4	Versión actualizada para el lanzamiento de la versión 2.5.4 del núcleo de Greengrass.
2.5.3	Versión actualizada para el lanzamiento de la versión 2.5.3 del núcleo de Greengrass.
2.5.2	Versión actualizada para el lanzamiento de la versión 2.5.2 del núcleo de Greengrass.
2.5.1	Versión actualizada para el lanzamiento de la versión 2.5.1 del núcleo de Greengrass.

Versión	Cambios
2.5.0	Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Agrega el nuevo parámetro de configuración `AuthorizedWindowsGroups` que puede especificar para autorizar a los grupos del sistema a utilizar la CLI de Greengrass en dispositivos Windows. • Agrega el parámetro `windowsUser` para las implementaciones locales. Puede usar este parámetro para especificar el usuario que se utilizará para ejecutar los componentes en un dispositivo principal de Windows.
2.4.0	Nuevas características • Suma compatibilidad con los límites de recursos del sistema. Al crear una implementación local, puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente puede utilizar en el dispositivo principal. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes o el comando create de la implementación.
2.3.0	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.2.0	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.0.5 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.0.4 del núcleo de Greengrass.
2.0.3	Versión inicial.

Detector de IP

El componente detector de IP (`aws.greengrass.clientdevices.IPDetector`) hace lo siguiente:

- Supervisa la información de conectividad de red del dispositivo principal de Greengrass. Esta información incluye los puntos de conexión de la red del dispositivo principal y el puerto en el que opera un agente de MQTT.
- Actualiza la información de conectividad del dispositivo principal en el servicio en la AWS IoT Greengrass nube.

Los dispositivos de cliente pueden usar la detección en la nube de Greengrass para recuperar la información de conectividad de sus dispositivos principales asociados. A continuación, los dispositivos de cliente pueden intentar conectarse a cada dispositivo principal hasta que se conecten correctamente.

> **Note**
>
> Los dispositivos de cliente son dispositivos IoT locales que se conectan a un dispositivo principal de Greengrass para enviar mensajes MQTT y datos para su procesamiento. Para obtener más información, consulte Interacción con dispositivos IoT locales.

El componente detector de IP reemplaza la información de conectividad existente de un dispositivo principal por la información que detecta. Como este componente elimina la información existente, puede utilizar el componente detector de IP o administrar manualmente la información de conectividad.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local

- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El rol de servicio de Greengrass debe estar asociado a sus permisos Cuenta de AWS y permitir los permisos `iot:GetThingShadow` `yiot:UpdateThingShadow`.
- La AWS IoT política del dispositivo principal debe permitir el `greengrass:UpdateConnectivityInfo` permiso. Para obtener más información, consulte

AWS IoT políticas para las operaciones del plano de datos y AWS IoT Política mínima de compatibilidad con los dispositivos cliente.

- Si configura el componente agente MQTT del dispositivo principal para que utilice un puerto que no sea el puerto 8883 predeterminado, debe usar el detector de IP versión 2.1.0 o posterior. Configúrelo para que indique el puerto en el que opera el agente.
- Si tiene una configuración de red compleja, es posible que el componente detector de IP no pueda identificar los puntos de conexión en los que los dispositivos de cliente se pueden conectar al dispositivo principal. Si el componente detector de IP no puede administrar los puntos de conexión, debe administrar manualmente los puntos de conexión del dispositivo principal. Por ejemplo, si el dispositivo principal está detrás de un enrutador que le reenvía el puerto de agente MQTT, debe especificar la dirección IP del enrutador como punto de conexión del dispositivo principal. Para obtener más información, consulte Administración de puntos de conexión del dispositivo principal.
- Se admite que el componente detector de IP se ejecute en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.2.3

En la siguiente tabla, se muestran las dependencias de la versión 2.2.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.17.0	Flexible

2.2.2

En la siguiente tabla, se muestran las dependencias de la versión 2.2.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 =2.2.0 <2.16.0	Flexible

2.2.1

En la siguiente tabla, se muestran las dependencias de la versión 2.2.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 =2.2.0 <2.15.0	Flexible

2.2.0

En la siguiente tabla, se muestran las dependencias de la versión 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.14.0	Flexible

2.1.8 – 2.1.9

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.8 y 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.13.0	Flexible

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.12.0	Flexible

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.11.0	Flexible

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.10.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.9.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.8.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.7.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.6.0	Flexible

2.1.0 and 2.0.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.0 y 2.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.5.0	Flexible

2.0.1

En la siguiente tabla, se muestran las dependencias de la versión 2.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.4.0	Flexible

2.0.0

En la siguiente tabla, se muestran las dependencias de la versión 2.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.3.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.2.x

`defaultPort`

(Opcional) El puerto del agente de MQTT para informar cuando este componente detecta direcciones IP. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883.

Valor predeterminado: `8883`

`includeIPv4LoopbackAddrs`

(Opcional) Puede activar esta opción para detectar y reportar direcciones de bucle invertido. IPv4 Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.

Valor predeterminado: `false`

`includeIPv4LinkLocalAddrs`

(Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.

Valor predeterminado: `false`

`includeIPv6LoopbackAddrs`

(Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv6 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción.

Valor predeterminado: `false`

`includeIPv6LinkLocalAddrs`

(Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv6 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción.

Valor predeterminado: `false`

`includeIPv4Addrs`

(Opcional) De forma predeterminada, la opción se establece en `true`. Puede activar esta opción para publicar IPv4 las direcciones que se encuentran en el dispositivo principal.

Valor predeterminado: `true`

`includeIPv6Addrs`

(Opcional) Puede activar esta opción para publicar IPv6 las direcciones que se encuentran en el dispositivo principal. Configure `includeIPv4Addrs` en `false` para usar esta opción.

Valor predeterminado: `false`

2.1.x

`defaultPort`

(Opcional) El puerto del agente de MQTT para informar cuando este componente detecta direcciones IP. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883.

Valor predeterminado: `8883`

`includeIPv4LoopbackAddrs`

(Opcional) Puede activar esta opción para detectar y reportar las direcciones de IPv4 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.

Valor predeterminado: `false`

`includeIPv4LinkLocalAddrs`

(Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.

Valor predeterminado: `false`

2.0.x

`includeIPv4LoopbackAddrs`

(Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv4 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.

Valor predeterminado: `false`

`includeIPv4LinkLocalAddrs`

(Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.

Valor predeterminado: `false`

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.2.3	Versión actualizada para el lanzamiento de la versión 2.16.0 de Greengrass nucleus.
2.2.2	Versión actualizada para el lanzamiento de la versión 2.15.0 de Greengrass nucleus.
2.2.1	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.2.0	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass. Nuevas características • Añade soporte para. IPv6 Ahora se puede utilizar IPv6 para mensajería local.
2.1.9	Mejoras y correcciones de errores • Ajusta el paso de IP adquirido para enviar solo los registros a nivel del registro de depuración.

Versión	Cambios
2.1.8	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.2	Mejoras y correcciones de errores • Mejora los mensajes de error que este componente registra en determinados escenarios. • Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.0	Mejoras • Agrega el parámetro `defaultPort` , que le permite utilizar un puerto de agente MQTT no predeterminado. • Actualizaciones para que los mensajes de registro sean más claros.
2.0.2	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.

Versión	Cambios
2.0.1	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.0	Versión inicial.

Firehose

El componente Firehose (`aws.greengrass.KinesisFirehose`) publica datos a través de las transmisiones de entrega de Amazon Data Firehose a destinos, como Amazon S3, Amazon Redshift y Amazon Service. OpenSearch Para obtener más información, consulte ¿Qué es Amazon Data Firehose? en la Guía para desarrolladores de Amazon Data Firehose.

Para publicar en un flujo de entrega de Kinesis con este componente, publique un mensaje en un tema al que se suscriba este componente. De forma predeterminada, este componente se suscribe a los temas de publicación/suscripción locales `kinesisfirehose/message` y `kinesisfirehose/message/binary/#`. Puede especificar otros temas, incluidos los de AWS IoT Core MQTT, al implementar este componente.

> **Note**
>
> Este componente proporciona una funcionalidad similar a la del conector Firehose de la V1. AWS IoT Greengrass Para obtener más información, consulte Conector de Firehose en la Guía para desarrolladores de AWS IoT Greengrass V1.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada

- Datos de salida
- Archivo de registro local
- Licencias
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de Lambda (`aws.greengrass.lambda`). El núcleo de Greengrass ejecuta la función de Lambda de este componente mediante el componente lanzador de Lambda.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- El rol de dispositivo de Greengrass debe permitir las acciones `firehose:PutRecord` y `firehose:PutRecordBatch`, tal como se muestra en la política de IAM en el ejemplo siguiente.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-name"
      ]
    }
  ]
}
```

Puede anular dinámicamente el flujo de entrega predeterminado en la carga útil del mensaje de entrada para este componente. Si su aplicación utiliza esta característica, la política de IAM debe incluir todos los flujos de destino como recursos. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

- Para recibir los datos de salida de este componente, debe combinar la siguiente actualización de configuración para el componente del enrutador de suscripción antiguo (`aws.greengrass.LegacySubscriptionRouter`) cuando implemente este componente. Esta configuración especifica el tema en el que este componente publica las respuestas.

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "component:aws.greengrass.KinesisFirehose",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-kinesisfirehose": {
      "id": "aws-greengrass-kinesisfirehose",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
kinesisfirehose:version",
      "subject": "kinesisfirehose/message/status",
      "target": "cloud"
    }
  }
}
```

- *region*Sustitúyalo por el Región de AWS que utilice.
- *version*Sustitúyala por la versión de la función Lambda que ejecuta este componente. Para encontrar la versión de la función de Lambda, debe ver la receta de la versión de este componente que desee implementar. Abra la página de detalles de este componente en la consola de AWS IoT Greengrass y busque el par clave-valor de la función de Lambda. Este par clave-valor contiene el nombre y la versión de la función de Lambda.

> **Important**
>
> Debe actualizar la versión de la función de Lambda en el enrutador de suscripción antiguo cada vez que implemente este componente. Esto garantiza que utilice la versión correcta de la función de Lambda para la versión del componente que implemente.

Para obtener más información, consulte Crear implementaciones.

- Se admite la ejecución del componente de Firehose en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.
  - El componente de Firehose debe tener una conectividad con `firehose.region.amazonaws.com` que tenga el punto de conexión de VPC de `com.amazonaws.region.kinesis-firehose`.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`firehose.` *`region`*`.amazonaws.com`	443	Sí	Carga de datos a Firehose.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.16.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 =2.0.0 <2.15.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.14.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.0.8 - 2.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.8 y 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Rígido
Lanzador de Lambda	>=1.0.0	Rígido
Tiempos de ejecución de Lambda	>=1.0.0	Flexible
Servicio de intercambio de token	>=1.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

> **Note**
>
> La configuración predeterminada de este componente incluye los parámetros de la función de Lambda. Le recomendamos que edite solo los siguientes parámetros para configurar este componente en sus dispositivos.

`lambdaParams`

Un objeto que contiene los parámetros de la función de Lambda de este componente. Este objeto contiene la siguiente información:

`EnvironmentVariables`

Un objeto que contiene los parámetros de la función de Lambda. Este objeto contiene la siguiente información:

`DEFAULT_DELIVERY_STREAM_ARN`

El ARN del flujo de entrega predeterminada de Firehose al que el componente envía los datos. Puede anular el flujo de destino con la propiedad `delivery_stream_arn` en la carga útil del mensajes de entrada.

> **Note**
>
> El rol del dispositivo principal debe permitir las acciones necesarias en todos los flujos de entrega de destino. Para obtener más información, consulte Requisitos.

`PUBLISH_INTERVAL`

(Opcional) La cantidad máxima de segundos que se deben esperar antes de que el componente publique los datos por lotes en Firehose. Para configurar el componente para que publique las métricas a medida que las reciba, es decir, sin procesamiento por lotes, especifique `0`.

Este valor puede ser de 900 segundos como máximo.

Valor predeterminado: 10 segundos

`DELIVERY_STREAM_QUEUE_SIZE`

(Opcional) La cantidad máxima de registros para retener en memoria antes de que el componente rechace los nuevos registros para el mismo flujo de entrega.

Este valor debe ser de al menos 2000 registros.

Valor predeterminado: 5000 registros

`containerMode`

(Opcional) El modo de almacenamiento en contenedores de este componente. Puede elegir entre las siguientes opciones:

- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.
- `GreengrassContainer`— El componente se ejecuta en un entorno de ejecución aislado dentro del contenedor. AWS IoT Greengrass

Valor predeterminado: `GreengrassContainer`

`containerParams`

(Opcional) Un objeto que contiene los parámetros de contenedor de este componente. El componente utiliza estos parámetros si se especifica `GreengrassContainer` para `containerMode`.

Este objeto contiene la siguiente información:

`memorySize`

(Opcional) La cantidad de memoria (en kilobytes) que se va a asignar al componente.

El valor predeterminado es 64 MB (65 535 KB).

`pubsubTopics`

(Opcional) Un objeto que contiene los temas a los que el componente se suscribe para recibir mensajes. Puede especificar cada tema y si el componente se suscribe a temas de MQTT AWS IoT Core o a temas locales publish/subscribe .

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`type`

(Opcional) El tipo de publish/subscribe mensajes que utiliza este componente para suscribirse a los mensajes. Puede elegir entre las siguientes opciones:

- `PUB_SUB` — Suscribirse a la mensajería de publicación/suscripción local. Si elige esta opción, el tema no podrá contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde un componente personalizado cuando especifique esta opción, consulte Publicar/suscribir mensajes locales.
- `IOT_CORE`— Suscríbase a los mensajes de AWS IoT Core MQTT. Si elige esta opción, el tema puede contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde componentes personalizados cuando especifique esta opción, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Valor predeterminado: `PUB_SUB`

`topic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `IotCore` para `type`, puede usar los comodines de MQTT (+ y #) en este tema.

Example Ejemplo: actualización de la combinación de configuraciones (modo en contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Ejemplo: actualización de la combinación de configuraciones (modo sin contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
```

```
  },
  "containerMode": "NoContainer"
}
```

Datos de entrada

Este componente acepta el contenido del flujo en los siguientes temas y envía el contenido al flujo de entrega de destino. El componente acepta dos tipos de datos de entrada:

- Datos JSON en el tema `kinesisfirehose/message`.
- Datos binarios en el tema `kinesisfirehose/message/binary/#`.

Tema predeterminado para los datos JSON (publicación/suscripción local): `kinesisfirehose/message`

El mensaje acepta las siguientes propiedades. Los mensajes de entrada deben tener un formato JSON válido.

`request`

Los datos para enviar a la transmisión de entrega y a la transmisión de entrega de destino, si no es la transmisión predeterminada.

Tipo: `object` que contiene la siguiente información:

`data`

Los datos que enviar a la transmisión de entrega.

Tipo: `string`

`delivery_stream_arn`

(Opcional) El ARN del flujo de entrega de destino de Firehose. Especifique esta propiedad para anular el flujo de entrega predeterminado.

Tipo: `string`

`id`

Un ID arbitrario para la solicitud. Use esta propiedad para asignar una solicitud de entrada a una respuesta de salida. Si especifica esta propiedad, el componente establece la propiedad `id` en el objeto de respuesta para este valor.

Tipo: `string`

Example Ejemplo de entrada

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-id:deliverystream/
stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Tema predeterminado para datos binarios (publicación/suscripción local): `kinesisfirehose/message/binary/#`

Consulte este tema para enviar un mensaje que contenga datos binarios. El componente no analiza los datos binarios. El componente transmite los datos tal cual.

Para asignar la solicitud de entrada a una respuesta de salida, sustituya el comodín # en el tema del mensaje con un ID de solicitud arbitrario. Por ejemplo, si publica un mensaje en `kinesisfirehose/message/binary/request123`, la propiedad `id` en el objeto de respuesta se establece en `request123`.

Si no desea asignar una solicitud a una respuesta, puede publicar sus mensajes en `kinesisfirehose/message/binary/`. Asegúrese de incluir la barra diagonal (`/`).

Datos de salida

Este componente publica las respuestas como datos de salida sobre el siguiente tema MQTT de forma predeterminada. Debe especificar este tema como parte de `subject` en la configuración del componente antiguo del enrutador de suscripciones. Para obtener más información sobre cómo suscribirse a los mensajes sobre este tema en sus componentes personalizados, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Tema predeterminado (AWS IoT Core MQTT): `kinesisfirehose/message/status`

Example Ejemplo de resultado de

La respuesta contiene el estado de cada registro de datos enviado en el lote.

```
{
  "response": [
    {
      "ErrorCode": "error",
      "ErrorMessage": "test error",
      "id": "request123",
      "status": "fail"
    },
    {
      "firehose_record_id": "xyz2",
      "id": "request456",
      "status": "success"
    },
    {
      "firehose_record_id": "xyz3",
      "id": "request890",
      "status": "success"
    }
  ]
}
```

> **Note**
>
> Si el componente detecta un error que se puede volver a intentar, como un error de conexión, volverá a intentar la publicación en el siguiente lote.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyalo por la ruta a la carpeta AWS IoT Greengrass raíz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
  ```

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- AWS SDK para Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.10	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.1.9	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.

Versión	Cambios
2.1.4	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Suma compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red y Permita que el dispositivo principal confíe en un proxy HTTPS.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Véase también

- ¿Qué es Amazon Data Firehose? en la Guía para desarrolladores de Amazon Data Firehose

Lanzador de Lambda

El componente Launcher Lambda (`aws.greengrass.LambdaLauncher`) inicia y detiene AWS Lambda las funciones en AWS IoT Greengrass los dispositivos principales. Este componente también configura cualquier organización de contenedores y ejecuta los procesos según los usuarios que especifique.

> **Note**
>
> Si implementa un componente de función de Lambda en un dispositivo principal, la implementación también incluye este componente. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Se admite la ejecución del componente lanzador de Lambda en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.0.11 – 2.0.13

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.11 a 2.0.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Administrador de Lambda	>=2.0.0 <2.4.0	Rígido

2.0.9 – 2.0.10

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.9 a 2.0.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Administrador de Lambda	>=2.0.0 <2.3.0	Rígido

2.0.4 - 2.0.8

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.4 a 2.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Administrador de Lambda	>=2.0.0 <2.2.0	Rígido

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Administrador de Lambda	>=2.0.3 <2.1.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyalo por la ruta a la carpeta AWS IoT Greengrass raíz y *lambdaFunctionComponentName* sustitúyalo por el nombre del componente de la función Lambda que lanza este componente.

  ```
  sudo tail -f /greengrass/v2/logs/lambdaFunctionComponentName.log
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.0.13	Mejoras y correcciones de errores Corrección de errores y mejoras generales.
2.0.12	Mejoras y correcciones de errores Solución de un problema por el que el lanzador Lambda podía generar un error si el proceso anterior no se detenía correctamente.
2.0.11	Soporte para el administrador de Lambda 2.3.0.
2.0.10	Mejoras y correcciones de errores • Corrección de errores y mejoras generales.
2.0.9	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.6	Mejoras de rendimiento generales y correcciones de errores.

Versión	Cambios
2.0.4	Mejoras y correcciones de errores • Solución de un problema por el que el componente `AddGroupOwner` no pasa correctamente al contenedor de función de Lambda.
2.0.3	Versión inicial.

Administrador de Lambda

El componente Lambda manager (`aws.greengrass.LambdaManager`) administra los elementos de trabajo y la comunicación entre procesos para AWS Lambda las funciones que se ejecutan en el dispositivo principal de Greengrass.

> **Note**
>
> Si implementa un componente de función de Lambda en un dispositivo principal, la implementación también incluye este componente. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

Temas

- Versiones
- Sistema operativo
- Tipo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Se admite la ejecución del componente administrador de Lambda en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia.

También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.3.7

En la siguiente tabla, se muestran las dependencias de la versión 2.3.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.17.0	Flexible

2.3.6

La siguiente tabla muestra las dependencias de la versión 2.3.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.16.0	Flexible

2.3.5

La siguiente tabla muestra las dependencias de la versión 2.3.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.15.0	Flexible

2.3.4

En la siguiente tabla, se muestran las dependencias de la versión 2.3.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.14.0	Flexible

2.3.2 and 2.3.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.2 y 2.3.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.2.10 and 2.3.1

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.10 y 2.3.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.2.8 and 2.2.9

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.8 y 2.2.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.2.7

En la siguiente tabla, se muestran las dependencias de la versión 2.2.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.2.6

En la siguiente tabla, se muestran las dependencias de la versión 2.2.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.2.5

En la siguiente tabla, se muestran las dependencias de la versión 2.2.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.2.4

En la siguiente tabla, se muestran las dependencias de la versión 2.2.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.2.1 - 2.2.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.1 a 2.2.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.2.0

En la siguiente tabla, se muestran las dependencias de la versión 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.6.0	Flexible

2.1.3 and 2.1.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.3 y 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`logHandlerMode`

> **Note**
>
> Solo para las versiones 2.3.0 y posteriores del administrador de Lambda

Se utiliza para elegir la implementación del administrador de registros de Lambda que se va a utilizar. Establezca el valor en `optimized` para usar menos subprocesos para leer los registros de Lambda.

`getResultTimeoutInSecond`

(Opcional) La cantidad máxima de tiempo en segundos que una función de Lambda puede ejecutarse antes de ponerse en espera.

Valor predeterminado: `60`

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

```
/greengrass/v2/logs/greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustitúyala por la */greengrass/v2* ruta a la carpeta raíz. AWS IoT Greengrass

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.3.7	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.3.6	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.3.5	Mejoras y correcciones de errores • Mejora el rendimiento al utilizar epoll en lugar de nio cuando está disponible.
2.3.4	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.3.3	Mejoras y correcciones de errores • Corrección de errores y mejoras generales.
2.3.2	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.3.1	Mejoras y correcciones de errores • Ajusta los niveles de registro para detectar determinados errores.
2.3.0	Nuevas características • El controlador de registros se ha optimizado para reducir la carga de la CPU. Utilice esta característica al establecer la opción de configuración `logHandlerMode` en `optimized` .

Versión	Cambios
	Mejoras y correcciones de errores • Ya no registra toda el rastro de pilas para `WorkQueueFullException`, lo que mejora los registros y el rendimiento. • Establece el tiempo de espera de apagado de Lambda de 15 a 300 segundos para evitar tiempos de espera de apagado. • Soluciona un problema que provocaba que las instancias de Lambdas bajo demanda no se reiniciaran después de cambiar la configuración.
2.2.11	Mejoras y correcciones de errores • Soluciona un problema por el que la LegacySubscriptionRouter configuración no se actualiza cuando cambia la configuración de Lambda.
2.2.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.2.9	Mejoras y correcciones de errores Soluciona un problema por el que el número de puerto estaba dañado debido a un reloj sesgado.
2.2.8	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.2.7	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.2.6	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.

Versión	Cambios
2.2.5	Nuevas características • Añade compatibilidad con los comodines de los temas MQTT en las fuentes de eventos en las que se suscribe a los mensajes locales. publish/subscribe Esta característica requiere la versión 2.6.0 o posterior del componente núcleo de Greengrass. • Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.2.4	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.2.3	Mejoras y correcciones de errores • Soluciona un problema por el que varias instancias de una función de Lambda comparten un único cgroup. Este componente usa cgroups para administrar el uso de recursos para las funciones de Lambda.
2.2.2	Mejoras y correcciones de errores • Soluciona un problema que provocaba que los componentes de la función de Lambda anclados se reiniciaran inesperadamente en determinadas situaciones.
2.2.1	Mejoras y correcciones de errores • Cambia las restricciones de la versión de dependencia del núcleo de Greengrass de este componente para solucionar un problema de resolución de dependencias.

Versión	Cambios
2.2.0	Mejoras y correcciones de errores • Soluciona un problema por el que las funciones de Lambda no podían escribir registros luego de un reinicio. • Soluciona un problema por el que el enrutador de suscripciones antiguo enviaba mensajes duplicados cuando había caracteres comodín en el tema. • Soluciona un problema por el que las funciones de Lambda no ancladas no podían utilizar la biblioteca de comunicación entre procesos (IPC) de Greengrass en el SDK para dispositivos con AWS IoT.
2.1.4	Mejoras y correcciones de errores • Corrige un problema que provocaba que las funciones de Lambda que utilizan tiempos de ejecución de Nodejs procesaran solo un mensaje. • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Tiempos de ejecución de Lambda

El componente de tiempos de ejecución de Lambda (`aws.greengrass.LambdaRuntimes`) proporciona los tiempos de ejecución que utilizan los dispositivos principales de Greengrass para ejecutar funciones de AWS Lambda.

> **Note**
>
> Si implementa un componente de función de Lambda en un dispositivo principal, la implementación también incluye este componente. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registro de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Se admite la ejecución del componente de tiempos de ejecución de Lambda en una VPC.

Dependencias

Este componente no tiene ninguna dependencia.

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente no genera registros.

Registro de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.0.9	Mejoras y correcciones de errores Soluciona una advertencia de sintaxis con Python 3.12
2.0.8	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.

Versión	Cambios
2.0.4	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Enrutador de suscripción antigua

El enrutador de suscripción antigua (`aws.greengrass.LegacySubscriptionRouter`) administra las suscripciones en el dispositivo principal de Greengrass. Las suscripciones son una función de la AWS IoT Greengrass versión 1 que define los temas que las funciones de Lambda pueden utilizar para la mensajería MQTT en un dispositivo principal. Para obtener más información, consulte las Suscripciones administradas en el flujo de trabajo de mensajería de MQTT en la Guía para desarrolladores de la V1 de AWS IoT Greengrass .

Puede usar este componente para habilitar las suscripciones de los componentes de conectores y los componentes de la función Lambda que utilizan el SDK de AWS IoT Greengrass Core.

> **Note**
>
> El componente de router de suscripción antiguo solo es necesario si la función Lambda utiliza la `publish()` función del SDK AWS IoT Greengrass principal. Si actualiza el código de la función Lambda para utilizar la interfaz de comunicación entre procesos (IPC) de la SDK para dispositivos con AWS IoT V2, no necesitará implementar el componente de router de suscripción heredado. Para obtener más información, consulte los siguientes servicios de comunicación entre procesos:
>
> - Publicar/suscribir mensajes locales
> - Publicar/suscribir mensajes MQTT AWS IoT Core

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos

- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- El enrutador de suscripción antigua es compatible para ejecutarse en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.15

En la siguiente tabla se enumeran las dependencias de la versión 2.1.15 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.17.0	Flexible

2.1.14

La siguiente tabla muestra las dependencias de la versión 2.1.14 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.16.0	Flexible

2.1.13

La siguiente tabla muestra las dependencias de la versión 2.1.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.15.0	Flexible

2.1.12

En la siguiente tabla, se muestran las dependencias de la versión 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.14.0	Flexible

2.1.11

En la siguiente tabla, se muestran las dependencias de la versión 2.1.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

v2.1.x

`subscriptions`

(Opcional) Las suscripciones que se van a habilitar en el dispositivo principal. Se trata de un objeto en el que cada clave es un identificador único y cada valor es un objeto que define la suscripción de ese conector. Debe configurar una suscripción al implementar un componente de conector V1 o una función Lambda que utilice el SDK AWS IoT Greengrass principal.

Cada objeto de suscripción contiene la siguiente información.

`id`

El identificador único de esta suscripción. Este identificador debe coincidir con la clave de este objeto de suscripción.

`source`

La función Lambda que usa el SDK AWS IoT Greengrass principal para publicar mensajes MQTT sobre los temas que especifique. `subject` Especifique uno de los siguientes valores:

- El nombre de un componente de la función de Lambda en el dispositivo principal. Especifique el nombre del componente con el prefijo `component:`, por ejemplo, **`component:com.example.HelloWorldLambda`**.
- El Nombre de recurso de Amazon (ARN) de la función de Lambda del dispositivo principal.

> **Important**
>
> Si la versión de la función de Lambda cambia, debe configurar la suscripción con la nueva versión de la función. De lo contrario, este componente no enrutará los mensajes hasta que la versión coincida con la suscripción.
> Debe especificar un Nombre de recurso de Amazon (ARN) que incluya la versión de la función que se va a importar. No puede utilizar alias de versión como `$LATEST`.

Para implementar una suscripción para un componente del conector V1, especifique el nombre del componente o el ARN de la función de Lambda del componente del conector.

`subject`

El tema o filtro de temas de MQTT en el que el origen y el destino pueden publicar y recibir mensajes. Este valor admite los caracteres comodín del tema + y #.

`target`

El destino que recibe los mensajes de MQTT sobre los temas que especifique en `subject`. La suscripción especifica que la `source` función publique los mensajes MQTT en AWS IoT Core o en una función Lambda del dispositivo principal. Especifique uno de los siguientes valores:

- `cloud`. La `source` función publica mensajes MQTT en. AWS IoT Core
- El nombre de un componente de la función de Lambda en el dispositivo principal. Especifique el nombre del componente con el prefijo `component:`, por ejemplo, **`component:com.example.HelloWorldLambda`**.
- El Nombre de recurso de Amazon (ARN) de la función de Lambda del dispositivo principal.

> **Important**
>
> Si la versión de la función de Lambda cambia, debe configurar la suscripción con la nueva versión de la función. De lo contrario, este componente no enrutará los mensajes hasta que la versión coincida con la suscripción.
> Debe especificar un Nombre de recurso de Amazon (ARN) que incluya la versión de la función que se va a importar. No puede utilizar alias de versión como `$LATEST`.

Predeterminado: sin suscripciones

Example Ejemplo de actualización de configuración (definición de una suscripción a AWS IoT Core)

El siguiente ejemplo especifica que el componente de la función `com.example.HelloWorldLambda` Lambda publica un mensaje MQTT AWS IoT Core en el tema. `hello/world`

```
{
  "subscriptions": {
```

```
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example Ejemplo de actualización de configuración (definición de una suscripción a otra función de Lambda)

En el siguiente ejemplo, se especifica que el componente de función de Lambda `com.example.HelloWorldLambda` publica mensajes de MQTT en el componente de función de Lambda `com.example.MessageRelay` en el tema `hello/world`.

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

(Opcional) Las suscripciones que se van a habilitar en el dispositivo principal. Se trata de un objeto en el que cada clave es un identificador único y cada valor es un objeto que define la suscripción de ese conector. Debe configurar una suscripción al implementar un componente de conector V1 o una función Lambda que utilice el SDK AWS IoT Greengrass principal.

Cada objeto de suscripción contiene la siguiente información.

`id`

El identificador único de esta suscripción. Este identificador debe coincidir con la clave de este objeto de suscripción.

`source`

La función Lambda que usa el SDK AWS IoT Greengrass principal para publicar mensajes MQTT sobre los temas que especifique. `subject` Especifique lo siguiente:

- El Nombre de recurso de Amazon (ARN) de la función de Lambda del dispositivo principal.

> **Important**
>
> Si la versión de la función de Lambda cambia, debe configurar la suscripción con la nueva versión de la función. De lo contrario, este componente no enrutará los mensajes hasta que la versión coincida con la suscripción.
> Debe especificar un Nombre de recurso de Amazon (ARN) que incluya la versión de la función que se va a importar. No puede utilizar alias de versión como `$LATEST`.

Para implementar una suscripción para un componente del conector V1, especifique el ARN de la función de Lambda del componente del conector.

`subject`

El tema o filtro de temas de MQTT en el que el origen y el destino pueden publicar y recibir mensajes. Este valor admite los caracteres comodín del tema + y #.

`target`

El destino que recibe los mensajes de MQTT sobre los temas que especifique en `subject`. La suscripción especifica que la `source` función publique los mensajes MQTT en AWS IoT Core o en una función Lambda del dispositivo principal. Especifique uno de los siguientes valores:

- `cloud`. La `source` función publica mensajes MQTT en. AWS IoT Core
- El Nombre de recurso de Amazon (ARN) de la función de Lambda del dispositivo principal.

> **Important**
>
> Si la versión de la función de Lambda cambia, debe configurar la suscripción con la nueva versión de la función. De lo contrario, este componente no enrutará los mensajes hasta que la versión coincida con la suscripción.

Debe especificar un Nombre de recurso de Amazon (ARN) que incluya la versión de la función que se va a importar. No puede utilizar alias de versión como `$LATEST`.

Predeterminado: sin suscripciones

Example Ejemplo de actualización de configuración (definición de una suscripción a AWS IoT Core)

El siguiente ejemplo especifica que la `Greengrass_HelloWorld` función publica un mensaje MQTT AWS IoT Core sobre el `hello/world` tema.

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
  }
}
```

Example Ejemplo de actualización de configuración (definición de una suscripción a otra función de Lambda)

En el siguiente ejemplo, se especifica que la función `Greengrass_HelloWorld` publica mensajes de MQTT en `Greengrass_MessageRelay` en el tema `hello/world`.

```
"subscriptions": {
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

Archivo de registro local

Este componente no genera registros.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.15	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.1.14	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.1.13	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.1.12	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.

Versión	Cambios
2.1.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Mejoras y correcciones de errores • Añade soporte para especificar los nombres de los componentes en lugar de ARNs para `source` y. `target` Si especifica el nombre de un componente para una suscripción, no es necesario volver a configurar la suscripción cada vez que cambie la versión de la función de Lambda.
2.0.3	Versión inicial.

Consola de depuración local

El componente de la consola de depuración local (`aws.greengrass.LocalDebugConsole`) proporciona un panel local que muestra información sobre los dispositivos AWS IoT Greengrass principales y sus componentes. Puede usar este panel para depurar su dispositivo principal y administrar los componentes locales.

> **Important**
>
> Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- De uso
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- Puede utilizar un nombre de usuario y una contraseña para iniciar sesión en el panel de control. Se le proporciona el nombre de usuario, que es `debug`. Debe usar la AWS IoT Greengrass CLI para crear una contraseña temporal que lo autentique con el panel de control de un dispositivo principal. Debe poder usar la AWS IoT Greengrass CLI para usar la consola de depuración local. Para obtener más información, consulte los requisitos de la CLI de Greengrass. Para obtener más información sobre cómo generar la contraseña e iniciar sesión, consulte Uso de los componentes de la consola de depuración local.
- Se admite la ejecución del componente de la consola de depuración local en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también implementa versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.4.6

En la siguiente tabla se enumeran las dependencias de la versión 2.4.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 <2.17.0	Rígido
CLI de Greengrass	>=2.10.0 =2.10.0 <2.17.0	Rígido

2.4.5

La siguiente tabla muestra las dependencias de la versión 2.4.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 <2.16.0	Rígido
CLI de Greengrass	>=2.10.0 =2.10.0 <2.16.0	Rígido

2.4.4

En la siguiente tabla, se muestran las dependencias de la versión 2.4.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 =2.10.0 <2.15.0	Rígido
CLI de Greengrass	>=2.10.0 =2.10.0 <2.15.0	Rígido

2.4.3

En la siguiente tabla, se muestran las dependencias de la versión 2.4.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 <2.14.0	Rígido
CLI de Greengrass	>=2.10.0 <2.14.0	Rígido

2.4.1 – 2.4.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.4.1 a 2.4.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 <2.13.0	Rígido
CLI de Greengrass	>=2.10.0 <2.13.0	Rígido

2.4.0

En la siguiente tabla, se muestran las dependencias de la versión 2.4.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 <2.12.0	Rígido
CLI de Greengrass	>=2.10.0 <2.12.0	Rígido

2.3.0 and 2.3.1

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.0 y 2.3.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.10.0 <2.12.0	Rígido
CLI de Greengrass	>=2.10.0 <2.12.0	Rígido

2.2.9

En la siguiente tabla, se muestran las dependencias de la versión 2.2.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.12.0	Rígido
CLI de Greengrass	>=2.1.0 <2.12.0	Rígido

2.2.8

En la siguiente tabla, se muestran las dependencias de la versión 2.2.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.11.0	Rígido
CLI de Greengrass	>=2.1.0 <2.11.0	Rígido

2.2.7

En la siguiente tabla, se muestran las dependencias de la versión 2.2.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.10.0	Rígido
CLI de Greengrass	>=2.1.0 <2.10.0	Rígido

2.2.6

En la siguiente tabla, se muestran las dependencias de la versión 2.2.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.9.0	Rígido
CLI de Greengrass	>=2.1.0 <2.9.0	Rígido

2.2.5

En la siguiente tabla, se muestran las dependencias de la versión 2.2.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.8.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
CLI de Greengrass	>=2.1.0 <2.8.0	Rígido

2.2.4

En la siguiente tabla, se muestran las dependencias de la versión 2.2.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.7.0	Rígido
CLI de Greengrass	>=2.1.0 <2.7.0	Rígido

2.2.3

En la siguiente tabla, se muestran las dependencias de la versión 2.2.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.6.0	Rígido
CLI de Greengrass	>=2.1.0 <2.6.0	Rígido

2.2.2

En la siguiente tabla, se muestran las dependencias de la versión 2.2.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.5.0	Rígido
CLI de Greengrass	>=2.1.0 <2.5.0	Rígido

2.2.1

En la siguiente tabla, se muestran las dependencias de la versión 2.2.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.4.0	Rígido
CLI de Greengrass	>=2.1.0 <2.4.0	Rígido

2.2.0

En la siguiente tabla, se muestran las dependencias de la versión 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.3.0	Rígido
CLI de Greengrass	>=2.1.0 <2.3.0	Rígido

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.2.0	Rígido
CLI de Greengrass	>=2.1.0 <2.2.0	Rígido

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible
CLI de Greengrass	>=2.0.3 <2.1.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

v2.1.x - v2.4.x

`httpsEnabled`

(Opcional) Puede habilitar la comunicación HTTPS para la consola de depuración local. Si habilita la comunicación HTTPS, la consola de depuración local crea un certificado autofirmado. Los navegadores web muestran advertencias de seguridad para los sitios web que utilizan certificados autofirmados, por lo que debe verificar el certificado manualmente. A continuación, puede omitir la advertencia. Para obtener más información, consulte De uso.

Valor predeterminado: `true`

`port`

(Opcional) El puerto en el que se va a proporcionar la consola de depuración local.

Valor predeterminado: `1441`

`websocketPort`

(Opcional) El puerto websocket que se utilizará en la consola de depuración local.

Valor predeterminado: `1442`

`bindHostname`

(Opcional) El nombre de host que se utilizará en la consola de depuración local.

Si ejecuta el software AWS IoT Greengrass principal en un contenedor de Docker, defina este parámetro en para poder abrir la consola de `0.0.0.0` depuración local fuera del contenedor de Docker.

Valor predeterminado: `localhost`

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo de configuración, se especifica abrir la consola de depuración local en puertos no predeterminados e inhabilitar HTTPS.

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

`port`

(Opcional) El puerto en el que se va a proporcionar la consola de depuración local.

Valor predeterminado: `1441`

`websocketPort`

(Opcional) El puerto websocket que se utilizará en la consola de depuración local.

Valor predeterminado: `1442`

`bindHostname`

(Opcional) El nombre de host que se utilizará en la consola de depuración local.

Si ejecuta el software AWS IoT Greengrass principal en un contenedor de Docker, defina este parámetro en para `0.0.0.0` poder abrir la consola de depuración local fuera del contenedor de Docker.

Valor predeterminado: `localhost`

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo de configuración, se especifica abrir la consola de depuración local en puertos no predeterminados.

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

De uso

Para usar la consola de depuración local, cree una sesión desde la CLI de Greengrass. Cuando crea una sesión, la CLI de Greengrass proporciona un nombre de usuario y una contraseña temporal que puede usar para iniciar sesión en la consola de depuración local.

Siga estas instrucciones para abrir la consola de depuración local en su dispositivo principal o en su computadora de desarrollo.

v2.1.x - v2.4.x

En las versiones 2.1.0 y posteriores, la consola de depuración local usa HTTPS de forma predeterminada. Cuando HTTPS está activado, la consola de depuración local crea un certificado autofirmado para proteger la conexión. Su navegador web muestra una advertencia de seguridad cuando abre la consola de depuración local debido a este certificado autofirmado. Cuando crea una sesión con la CLI de Greengrass, el resultado incluye las huellas digitales del certificado para que pueda comprobar que el certificado es legítimo y que la conexión es segura.

Puede deshabilitar HTTPS. Para obtener más información, consulte Configuración de la consola de depuración local.

Cómo abrir la consola de depuración local

1. (Opcional) Para ver la consola de depuración local en su computadora de desarrollo, puede reenviar el puerto de la consola a través de SSH. Sin embargo, primero debe habilitar la opción `AllowTcpForwarding` en el archivo de configuración SSH de su dispositivo principal. Esta opción está habilitada de forma predeterminada. Ejecute el siguiente comando en su computadora de desarrollo para ver el panel de control en `localhost:1441` en su computadora de desarrollo.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > Puede cambiar los puertos predeterminados entre 1441 y 1442. Para obtener más información, consulte Configuración de la consola de depuración local.

2. Cree una sesión para usar la consola de depuración local. Cuando crea una sesión, genera una contraseña que se utiliza para autenticarse. La consola de depuración local requiere una contraseña para aumentar la seguridad, ya que puede utilizar este componente para ver información importante y realizar operaciones en el dispositivo principal. La consola de depuración local también crea un certificado para proteger la conexión si se habilita HTTPS en la configuración del componente. HTTPS está habilitado de forma predeterminada.

   Utilice la AWS IoT Greengrass CLI para crear la sesión. Este comando genera una contraseña aleatoria de 43 caracteres que caduca a las 8 horas. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass V2 raíz.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

   El resultado del comando es similar al del siguiente ejemplo si ha configurado la consola de depuración local para que utilice HTTPS. Use las huellas digitales del certificado para comprobar que la conexión es segura cuando abra la consola de depuración local.

   ```
   Username: debug
   Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
   Password expires at: 2021-04-01T17:01:43.921999931-07:00
   The local debug console is configured to use TLS security. The certificate is
    self-signed so you will need to bypass your web browser's security warnings to
    open the console.
   Before you bypass the security warning, verify that the certificate fingerprint
    matches the following fingerprints.
   SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
    96 DA A6 CC B1 D2 C4 1B
   SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
   ```

   El componente de vista de depuración crea una sesión que dura 8 horas. Después de eso, debe generar una nueva contraseña para volver a ver la consola de depuración local.

3. Abra e inicie sesión en el panel de control. Vea el panel de control en su dispositivo principal de Greengrass o en su computadora de desarrollo si reenvía el puerto a través de SSH. Realice una de las siguientes acciones:

   - Si habilitó HTTPS en la consola de depuración local, que es la configuración predeterminada, haga lo siguiente:

     a. Abra `https://localhost:1441` en su dispositivo principal o en su computadora de desarrollo si reenvió el puerto a través de SSH.

        Es posible que su navegador muestre una advertencia de seguridad sobre un certificado de seguridad no válido.

     b. Si su navegador muestra una advertencia de seguridad, compruebe que el certificado sea legítimo y omita la advertencia de seguridad. Haga lo siguiente:

        i. Busque la huella digital SHA-256 o SHA-1 del certificado y compruebe que coincida con la huella digital SHA-256 o SHA-1 que el comando `get-debug-password` imprimió anteriormente. Es posible que su navegador proporcione una huella digital o ambas. Consulte la documentación de su navegador para ver el certificado y sus huellas digitales. En algunos navegadores, la huella digital del certificado se denomina huella digital.

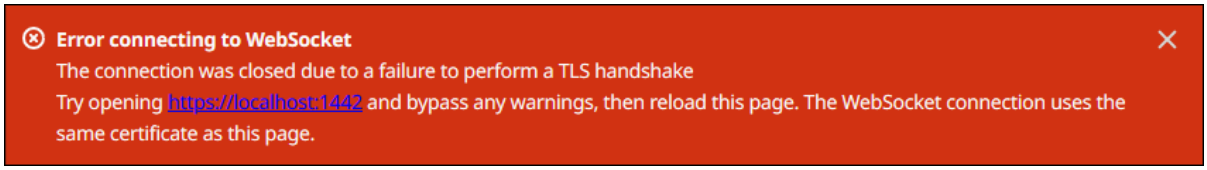
        > **Note**
        >
        > Si la huella digital del certificado no coincide, vaya a Step 2 para crear una nueva sesión. Si la huella digital del certificado sigue sin coincidir, es posible que la conexión no sea segura.

        ii. Si la huella digital del certificado coincide, omita la advertencia de seguridad del navegador para abrir la consola de depuración local. Consulte la documentación del navegador para omitir la advertencia de seguridad del navegador.

     c. Inicie sesión en el sitio web con el nombre de usuario y la contraseña que el comando `get-debug-password` imprimió anteriormente.

        Se abre la consola de depuración local.

d. Si la consola de depuración local muestra un error que indica que no se puede conectar a ella WebSocket debido a un error en el protocolo de enlace TLS, debes omitir la advertencia de seguridad autofirmada de la URL. WebSocket



Haga lo siguiente:

i. Abra `https://localhost:1442` en el mismo navegador en el que abrió la consola de depuración local.

ii. Compruebe el certificado y omita la advertencia de seguridad.

Es posible que su navegador muestre una página HTTP 404 después de omitir la advertencia.

iii. Abra `https://localhost:1441` de nuevo.

La consola de depuración local muestra información sobre el dispositivo principal.

- Si ha deshabilitado HTTPS en la consola de depuración local, haga lo siguiente:

a. Abra `http://localhost:1441` en su dispositivo principal o ábralo en su computadora de desarrollo si reenvió el puerto a través de SSH.

b. Inicie sesión en el sitio web con el nombre de usuario y la contraseña que el comando `get-debug-password` imprimió anteriormente.

Se abre la consola de depuración local.

v2.0.x

Cómo abrir la consola de depuración local

1. (Opcional) Para ver la consola de depuración local en su computadora de desarrollo, puede reenviar el puerto de la consola a través de SSH. Sin embargo, primero debe habilitar la opción `AllowTcpForwarding` en el archivo de configuración SSH de su dispositivo principal. Esta opción está habilitada de forma predeterminada. Ejecute el siguiente comando

en su computadora de desarrollo para ver el panel de control en `localhost:1441` en su computadora de desarrollo.

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

> **Note**
>
> Puede cambiar los puertos predeterminados entre 1441 y 1442. Para obtener más información, consulte Configuración de la consola de depuración local.

2. Cree una sesión para usar la consola de depuración local. Cuando crea una sesión, genera una contraseña que se utiliza para autenticarse. La consola de depuración local requiere una contraseña para aumentar la seguridad, ya que puede utilizar este componente para ver información importante y realizar operaciones en el dispositivo principal.

   Utilice la AWS IoT Greengrass CLI para crear la sesión. Este comando genera una contraseña aleatoria de 43 caracteres que caduca a las 8 horas. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass V2 raíz.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

   El resultado del comando tendrá un aspecto similar al siguiente ejemplo.

   ```
   Username: debug
   Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
   Password will expire at: 2021-04-01T17:01:43.921999931-07:00
   ```

   El componente de vista de depuración crea una sesión que dura 4 horas y, a continuación, debe generar una nueva contraseña para volver a ver la consola de depuración local.

3. Abra `http://localhost:1441` en su dispositivo principal o ábralo en su computadora de desarrollo si reenvió el puerto a través de SSH.
4. Inicie sesión en el sitio web con el nombre de usuario y la contraseña que el comando `get-debug-password` imprimió anteriormente.

   Se abre la consola de depuración local.

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.4.6	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.4.5	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.4.4	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.4.3	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass. Mejoras y correcciones de errores • Corrige un problema que mostraba incorrectamente `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH` en megabits por segundo (Mbps) en lugar de en bytes por segundo (Bps).
2.4.2	Mejoras y correcciones de errores • Corrección de errores y mejoras generales.
2.4.1	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.4.0	Nuevas características • Agrega la consola de depuración del Administrador de flujos.
2.3.1	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.3.0	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass. Nuevas características • Incluye un cliente de PubSub depuración AWS IoT Core MQTT.
2.2.7	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.2.6	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.

Versión	Cambios
2.2.5	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.2.4	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.2.3	Mejoras y correcciones de errores • Corrige un problema que impedía el arranque cuando el componente no podía descifrar el almacén de claves que contiene la clave privada SSL. • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.2.2	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.2.1	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.2.0	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Utiliza HTTPS para proteger la conexión a la consola de depuración local. HTTPS está habilitado de forma predeterminada. Mejoras y correcciones de errores • Puede descartar los mensajes de la barra flash en el editor de configura ción.
2.0.3	Versión inicial.

Administrador de registros

> **Warning**
>
> Se recomienda actualizar a la versión 2.3.5 o posterior del administrador de registros. La versión 2.3.5 optimiza la escritura de la configuración del administrador de registros, lo que reduce las operaciones de E/A y mejora la velocidad de carga de los registros, el rendimiento general del dispositivo y, posiblemente, prolonga su vida útil.

El componente gestor de registros (`aws.greengrass.LogManager`) carga los registros de los dispositivos AWS IoT Greengrass principales a Amazon CloudWatch Logs. Puede cargar registros desde el núcleo de Greengrass, otros componentes de Greengrass y otras aplicaciones y servicios que no sean componentes de Greengrass. Para obtener más información sobre cómo supervisar los CloudWatch registros en Logs y en el sistema de archivos local, consulteSupervisión de los registros de AWS IoT Greengrass.

Cuando se utiliza el componente gestor de registros para escribir en los CloudWatch registros, se tienen en cuenta las siguientes consideraciones:

- Demoras de registro

  La versión 2.2.8 (y anteriores) del componente administrador de registros procesa y carga los registros únicamente a partir de archivos de registro rotados. De forma predeterminada, el software AWS IoT Greengrass Core rota los archivos de registro cada hora o después de que ocupen 1024 KB. Como resultado, el componente del administrador de registros carga los registros solo después de que el software AWS IoT Greengrass Core o un componente de Greengrass hayan escrito registros con un valor superior a 1024 KB. Puede configurar un límite de tamaño de archivo de registro inferior para que los archivos de registro roten con mayor frecuencia. Esto hace que el componente del administrador de registros cargue registros en Logs con CloudWatch más frecuencia.

  La versión 2.3.0 (y posteriores) del componente administrador de registros procesa y carga todos los registros. Cuando escribe un registro nuevo, la versión 2.3.0 (y posteriores) del administrador de registros procesa y carga directamente el archivo de registro activo en lugar de esperar a que se rote. Esto significa que puede ver el nuevo registro en 5 minutos o menos.

  El componente administrador de registros carga nuevos registros periódicamente. De forma predeterminada, el componente del administrador de registros carga nuevos registros cada

5 minutos. Puede configurar un intervalo de carga más bajo, de modo que el componente del administrador de registros cargue los registros en CloudWatch Logs con más frecuencia configurando el`periodicUploadIntervalSec`. Para obtener más información acerca de cómo configurar este intervalo periódico, consulte Configuración.

Los registros se pueden cargar prácticamente en tiempo real desde el mismo sistema de archivos de Greengrass. Si necesita observar los registros en tiempo real, considere usar los registros del sistema de archivos.

> **Note**
>
> Si utiliza distintos sistemas de archivos para escribir los registros, el administrador de registros vuelve al comportamiento de las versiones 2.2.8 y anteriores del componente administrador de registros. Para obtener información sobre cómo acceder a los registros del sistema de archivos, consulte Acceder a los registros del sistema de archivos.

- Desfase del reloj

  El componente gestor de registros utiliza el proceso de firma estándar de la versión 4 de Signature para crear solicitudes de API a CloudWatch los registros. Si la hora del sistema en un dispositivo principal está desincronizada durante más de 15 minutos, CloudWatch Logs rechaza las solicitudes. Para obtener más información, consulte Proceso de firma Signature Version 4 en la Referencia general de AWS.

Para obtener información sobre los grupos de registros y los flujos de registros a los que este componente carga los registros, consulte De uso.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- De uso
- Archivo de registro local

- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El rol de dispositivo de Greengrass debe permitir las acciones `logs:CreateLogGroup`, `logs:CreateLogStream`, `logs:PutLogEvents` y `logs:DescribeLogStreams` como se muestra en el siguiente ejemplo de política de IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

> **Note**
>
> La función de dispositivo Greengrass que se crea al instalar el software AWS IoT Greengrass Core incluye los permisos de esta política de ejemplo de forma predeterminada.

Para obtener más información, consulte Uso de políticas basadas en la identidad (políticas de IAM) para CloudWatch los registros en la Guía del usuario de Amazon CloudWatch Logs.

- Se admite la ejecución del componente administrador de registros en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.
  - El componente del administrador de registros debe tener una conectividad con `logs.region.amazonaws.com` que tenga el punto de conexión de VPC de `com.amazonaws.us-east-1.logs`.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`logs.region.amazonaws.com`	443	No	Es obligatorio si escribes registros en Logs. CloudWatch

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.3.11

En la siguiente tabla se enumeran las dependencias de la versión 2.3.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.17.0	Flexible

2.3.10

La siguiente tabla muestra las dependencias de la versión 2.3.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.16.0	Flexible

2.3.9

En la siguiente tabla, se muestran las dependencias de la versión 2.3.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 =2.1.0 <2.15.0	Flexible

2.3.8

En la siguiente tabla, se muestran las dependencias de la versión 2.3.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.14.0	Flexible

2.3.7

En la siguiente tabla, se muestran las dependencias de la versión 2.3.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.13.0	Flexible

2.3.5 and 2.3.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.5 y 2.3.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.12.0	Flexible

2.3.3 – 2.3.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.3 a 2.3.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.11.0	Flexible

2.2.8 – 2.3.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.8 a 2.3.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.10.0	Flexible

2.2.7

En la siguiente tabla, se muestran las dependencias de la versión 2.2.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.9.0	Flexible

2.2.6

En la siguiente tabla, se muestran las dependencias de la versión 2.2.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.8.0	Flexible

2.2.5

En la siguiente tabla, se muestran las dependencias de la versión 2.2.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.7.0	Flexible

2.2.1 - 2.2.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.1 a 2.2.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.6.0	Flexible

2.1.3 and 2.2.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.3 y 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.5.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.4.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.3.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.1.0 <2.2.0	Flexible

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

v2.3.10

`logsUploaderConfiguration`

(Opcional) La configuración de los registros que carga el componente administrador de registros. Este objeto contiene la siguiente información:

`systemLogsConfiguration`

(Opcional) La configuración de los registros del sistema del software AWS IoT Greengrass principal, que incluyen los registros del núcleo y los componentes del complemento de

Greengrass. Especifique esta configuración para permitir que el componente administrador de registros administre los registros del sistema. Este objeto contiene la siguiente información:

`uploadToCloudWatch`

(Opcional) Puede cargar los registros del sistema en CloudWatch Logs.

Valor predeterminado: `false`

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si configura el componente núcleo de Greengrass para generar registros en formato JSON. Para habilitar los registros en formato JSON, especifique `JSON` para el parámetro de formato de registro (`logging.format`).

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de los archivos de registro del sistema Greengrass en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro del sistema Greengrass supera este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro del sistema Greengrass más antiguos.

Este parámetro equivale al parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño máximo total del registro del sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`componentLogsConfigurationMap`

(Opcional) Un mapa de las configuraciones de registro de los componentes del dispositivo principal. Cada objeto `componentName` de este mapa define la configuración de registro del componente o la aplicación. El componente gestor de registros carga estos registros de componentes en Logs. CloudWatch

> **Important**
>
> Recomendamos utilizar una única clave de configuración por componente. Solo debe dirigirse a un grupo de archivos que solo tengan un archivo de registro en el que se esté escribiendo activamente cuando use `logFileRegex`. Si no se sigue esta recomendación, es posible que se carguen registros duplicados en ellos. CloudWatch Si se dirige a varios archivos de registro activos con una sola expresión regular, le recomendamos que actualice a la versión 2.3.1 o posterior del administrador de registros y que considere la posibilidad de cambiar la configuración con la configuración de ejemplo.

> **Note**
>
> Si está actualizando desde una versión del administrador de registros anterior a la versión 2.2.0, puede seguir utilizando la lista `componentLogsConfiguration` en

lugar de `componentLogsConfigurationMap`. No obstante, le recomendamos que utilice el formato de mapa para poder utilizar actualizaciones de combinación y restablecimiento para modificar configuraciones de componentes específicos. Para obtener información sobre el parámetro `componentLogsConfiguration`, consulte los parámetros de configuración de la versión 2.1.x de este componente.

`componentName`

La configuración de registro del componente o la aplicación *`componentName`* de esta configuración de registro. Puede especificar el nombre de un componente de Greengrass u otro valor para identificar este grupo de registro.

Cada objeto contiene la siguiente información:

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo solo se aplica si los registros de este componente utilizan un formato JSON específico, que puedes encontrar en el repositorio del módulo de AWS IoT Greengrass registro de GitHub.

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de todos los archivos de registro de este componente en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro de este componente supere este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de este componente.

Este parámetro está relacionado con el parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT

Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para este componente.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`logFileDirectoryPath`

(Opcional) La ruta a la carpeta que contiene los archivos de registro de este componente.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Predeterminado: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Opcional) Expresión habitual que especifica el formato de nombre del archivo de registro que utiliza el componente o la aplicación. El componente administrador de registro utiliza esta expresión habitual para identificar los archivos de registro de la carpeta situada en `logFileDirectoryPath`.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Si su componente o aplicación rota los archivos de registro, especifique una expresión habitual que coincida con los nombres de los archivos de registro rotados. Por ejemplo, puede especificar **`hello_world\\\\w*.log`** para cargar los registros de una aplicación de Hello World. El patrón `\\\\w*` coincide con cero o más caracteres de palabra, lo que incluye caracteres alfanuméricos y guiones bajos. Esta expresión habitual hace coincidir los archivos de registro con y sin marcas temporales en su nombre. En este ejemplo, el administrador de registro carga los siguientes archivos de registro:

- `hello_world.log`: el archivo de registro más reciente de la aplicación Hello World.
- `hello_world_2020_12_15_17_0.log`: un archivo de registro más antiguo de la aplicación Hello World.

Predeterminado:*componentName*`\\\\w*.log`, donde *componentName* es el nombre del componente para esta configuración de registro.

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en CloudWatch Logs.

Valor predeterminado: `false`

`multiLineStartPattern`

(Opcional) Expresión habitual que identifica cuándo un mensaje de registro de una línea nueva es un mensaje de registro nuevo. Si la expresión regular no coincide con la nueva línea, el componente administrador de registros anexa la nueva línea al mensaje de registro de la línea anterior.

De forma predeterminada, el componente administrador de registros comprueba si la línea comienza con un carácter de espacio en blanco, como un tabulador o un espacio. Si no es así, el administrador de registros trata esa línea como un nuevo mensaje de registro. De lo contrario, anexa esa línea al mensaje de registro actual. Este comportamiento garantiza que el componente administrador de registros no divida los mensajes que ocupan varias líneas, como los rastreos de pila.

`periodicUploadIntervalSec`

(Opcional) El periodo en segundos durante el que el componente administrador de registros comprueba si hay nuevos archivos de registro para cargar.

Valor predeterminado: `300` (5 minutos)

Mínimo: `0.000001` (1 microsegundo)

`updateToTlogIntervalSec`

(Opcional) El periodo en segundos con el que el núcleo escribe los detalles de los eventos de carga de registros de Amazon CloudWatch Events en el registro de

transacción local (`config.tlog`). Los valores predeterminados se especifican en `periodicUploadIntervalSec`. Puede modificar este parámetro para aumentar el intervalo de escritura.

Valor predeterminado: `periodicUploadIntervalSec`

Mínimo: `periodicUploadIntervalSec`

`deprecatedVersionSupport`

Indica si el administrador de registros debe utilizar las mejoras de velocidad de registro introducidas en la versión 2.3.5 del administrador de registros. Establezca el valor en `false` para utilizar las mejoras.

Si establece este valor `false` cuando actualice desde la versión 2.3.1 del administrador de registros o de una versión anterior, es posible que se carguen entradas de registro duplicadas.

El valor predeterminado es `true`.

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica cargar los registros del sistema y los registros de los `com.example.HelloWorld` componentes en CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
```

```
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example Ejemplo: configuración para cargar varios archivos de registro activos mediante la versión 2.3.1 del administrador de registros

El siguiente ejemplo de configuración es el ejemplo recomendado si desea dirigirse a varios archivos de registro activos. En este ejemplo de configuración se especifican los archivos de registro activos en los que desea cargarlos CloudWatch. Si utiliza este ejemplo de configuración, la configuración también cargará todos los archivos rotados que coincidan con `logFileRegex`. Este ejemplo de configuración es compatible con el administrador de registros versión 2.3.1.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.6 – v2.3.9

`logsUploaderConfiguration`

(Opcional) La configuración de los registros que carga el componente administrador de registros. Este objeto contiene la siguiente información:

`systemLogsConfiguration`

(Opcional) La configuración de los registros del sistema del software AWS IoT Greengrass principal, que incluyen los registros del núcleo y los componentes del complemento de Greengrass. Especifique esta configuración para permitir que el componente administrador

de registros administre los registros del sistema. Este objeto contiene la siguiente información:

`uploadToCloudWatch`

(Opcional) Puede cargar los registros del sistema en CloudWatch Logs.

Valor predeterminado: `false`

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si configura el componente núcleo de Greengrass para generar registros en formato JSON. Para habilitar los registros en formato JSON, especifique `JSON` para el parámetro de formato de registro (`logging.format`).

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de los archivos de registro del sistema Greengrass en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro del sistema Greengrass supera este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro del sistema Greengrass más antiguos.

Este parámetro equivale al parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño máximo total del registro del sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`componentLogsConfigurationMap`

(Opcional) Un mapa de las configuraciones de registro de los componentes del dispositivo principal. Cada objeto `componentName` de este mapa define la configuración de registro del componente o la aplicación. El componente gestor de registros carga estos registros de componentes en Logs. CloudWatch

> **Important**
>
> Recomendamos utilizar una única clave de configuración por componente. Solo debe dirigirse a un grupo de archivos que solo tengan un archivo de registro en el que se esté escribiendo activamente cuando use `logFileRegex`. Si no se sigue esta recomendación, es posible que se carguen registros duplicados en ellos. CloudWatch Si se dirige a varios archivos de registro activos con una sola expresión regular, le recomendamos que actualice a la versión 2.3.1 o posterior del administrador de registros y que considere la posibilidad de cambiar la configuración con la configuración de ejemplo.

> **Note**
>
> Si está actualizando desde una versión del administrador de registros anterior a la versión 2.2.0, puede seguir utilizando la lista `componentLogsConfiguration` en lugar de `componentLogsConfigurationMap`. No obstante, le recomendamos que utilice el formato de mapa para poder utilizar actualizaciones de combinación y restablecimiento para modificar configuraciones de componentes específicos.

Para obtener información sobre el parámetro `componentLogsConfiguration`, consulte los parámetros de configuración de la versión 2.1.x de este componente.

`componentName`

La configuración de registro del componente o la aplicación *`componentName`* de esta configuración de registro. Puede especificar el nombre de un componente de Greengrass u otro valor para identificar este grupo de registro.

Cada objeto contiene la siguiente información:

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo solo se aplica si los registros de este componente utilizan un formato JSON específico, que puedes encontrar en el repositorio del módulo de AWS IoT Greengrass registro de GitHub.

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de todos los archivos de registro de este componente en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro de este componente supere este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de este componente.

Este parámetro está relacionado con el parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para este componente.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`logFileDirectoryPath`

(Opcional) La ruta a la carpeta que contiene los archivos de registro de este componente.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Predeterminado: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Opcional) Expresión habitual que especifica el formato de nombre del archivo de registro que utiliza el componente o la aplicación. El componente administrador de registro utiliza esta expresión habitual para identificar los archivos de registro de la carpeta situada en `logFileDirectoryPath`.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Si su componente o aplicación rota los archivos de registro, especifique una expresión habitual que coincida con los nombres de los archivos de registro rotados. Por ejemplo, puede especificar **`hello_world\\\\w*.log`** para cargar los registros de una aplicación de Hello World. El patrón `\\\\w*` coincide con cero o más caracteres de palabra, lo que incluye caracteres alfanuméricos y guiones bajos. Esta expresión habitual hace coincidir los archivos de registro con y sin marcas temporales en su nombre. En este ejemplo, el administrador de registro carga los siguientes archivos de registro:

- `hello_world.log`: el archivo de registro más reciente de la aplicación Hello World.

- `hello_world_2020_12_15_17_0.log`: un archivo de registro más antiguo de la aplicación Hello World.

Predeterminado:*componentName*`\\\\w*.log`, donde *componentName* es el nombre del componente para esta configuración de registro.

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en CloudWatch Logs.

Valor predeterminado: `false`

`multiLineStartPattern`

(Opcional) Expresión habitual que identifica cuándo un mensaje de registro de una línea nueva es un mensaje de registro nuevo. Si la expresión regular no coincide con la nueva línea, el componente administrador de registros anexa la nueva línea al mensaje de registro de la línea anterior.

De forma predeterminada, el componente administrador de registros comprueba si la línea comienza con un carácter de espacio en blanco, como un tabulador o un espacio. Si no es así, el administrador de registros trata esa línea como un nuevo mensaje de registro. De lo contrario, anexa esa línea al mensaje de registro actual. Este comportamiento garantiza que el componente administrador de registros no divida los mensajes que ocupan varias líneas, como los rastreos de pila.

`periodicUploadIntervalSec`

(Opcional) El periodo en segundos durante el que el componente administrador de registros comprueba si hay nuevos archivos de registro para cargar.

Valor predeterminado: `300` (5 minutos)

Mínimo: `0.000001` (1 microsegundo)

`deprecatedVersionSupport`

Indica si el administrador de registros debe utilizar las mejoras de velocidad de registro introducidas en la versión 2.3.5 del administrador de registros. Establezca el valor en `false` para utilizar las mejoras.

Si establece este valor `false` cuando actualice desde la versión 2.3.1 del administrador de registros o de una versión anterior, es posible que se carguen entradas de registro duplicadas.

El valor predeterminado es `true`.

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica cargar los registros del sistema y los registros de los `com.example.HelloWorld` componentes en CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example Ejemplo: configuración para cargar varios archivos de registro activos mediante la versión 2.3.1 del administrador de registros

El siguiente ejemplo de configuración es el ejemplo recomendado si desea dirigirse a varios archivos de registro activos. En este ejemplo de configuración se especifican los archivos de registro activos en los que desea cargarlos CloudWatch. Si utiliza este ejemplo de configuración, la configuración también cargará todos los archivos rotados que coincidan con `logFileRegex`. Este ejemplo de configuración es compatible con el administrador de registros versión 2.3.1.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
```

```
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.0 – 2.3.5

`logsUploaderConfiguration`

(Opcional) La configuración de los registros que carga el componente administrador de registros. Este objeto contiene la siguiente información:

`systemLogsConfiguration`

(Opcional) La configuración de los registros del sistema del software AWS IoT Greengrass principal, que incluyen los registros del núcleo y los componentes del complemento de Greengrass. Especifique esta configuración para permitir que el componente administrador de registros administre los registros del sistema. Este objeto contiene la siguiente información:

`uploadToCloudWatch`

(Opcional) Puede cargar los registros del sistema en CloudWatch Logs.

Valor predeterminado: `false`

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si configura el componente núcleo de Greengrass para generar registros en formato JSON. Para habilitar los registros en formato JSON, especifique `JSON` para el parámetro de formato de registro (`logging.format`).

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de los archivos de registro del sistema Greengrass en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro del sistema Greengrass supera este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro del sistema Greengrass más antiguos.

Este parámetro equivale al parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño máximo total del registro del sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`componentLogsConfigurationMap`

(Opcional) Un mapa de las configuraciones de registro de los componentes del dispositivo principal. Cada objeto `componentName` de este mapa define la configuración de registro

del componente o la aplicación. El componente gestor de registros carga estos registros de componentes en Logs. CloudWatch

> **Important**
>
> Recomendamos utilizar una única clave de configuración por componente. Solo debe dirigirse a un grupo de archivos que solo tengan un archivo de registro en el que se esté escribiendo activamente cuando use `logFileRegex`. Si no se sigue esta recomendación, es posible que se carguen registros duplicados en ellos. CloudWatch Si se dirige a varios archivos de registro activos con una sola expresión regular, le recomendamos que actualice a la versión 2.3.1 del administrador de registros y que considere la posibilidad de cambiar la configuración con la configuración de ejemplo.

> **Note**
>
> Si está actualizando desde una versión del administrador de registros anterior a la versión 2.2.0, puede seguir utilizando la lista `componentLogsConfiguration` en lugar de `componentLogsConfigurationMap`. No obstante, le recomendamos que utilice el formato de mapa para poder utilizar actualizaciones de combinación y restablecimiento para modificar configuraciones de componentes específicos. Para obtener información sobre el parámetro `componentLogsConfiguration`, consulte los parámetros de configuración de la versión 2.1.x de este componente.

`componentName`

La configuración de registro del componente o la aplicación *`componentName`* de esta configuración de registro. Puede especificar el nombre de un componente de Greengrass u otro valor para identificar este grupo de registro.

Cada objeto contiene la siguiente información:

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo solo se aplica si los registros de este componente utilizan un formato

JSON específico, que puedes encontrar en el repositorio del módulo de AWS IoT Greengrass registro de GitHub.

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de todos los archivos de registro de este componente en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro de este componente supere este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de este componente.

Este parámetro está relacionado con el parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para este componente.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`logFileDirectoryPath`

(Opcional) La ruta a la carpeta que contiene los archivos de registro de este componente.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Predeterminado: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Opcional) Expresión habitual que especifica el formato de nombre del archivo de registro que utiliza el componente o la aplicación. El componente administrador de registro utiliza esta expresión habitual para identificar los archivos de registro de la carpeta situada en `logFileDirectoryPath`.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Si su componente o aplicación rota los archivos de registro, especifique una expresión habitual que coincida con los nombres de los archivos de registro rotados. Por ejemplo, puede especificar **`hello_world\\\\w*.log`** para cargar los registros de una aplicación de Hello World. El patrón `\\\\w*` coincide con cero o más caracteres de palabra, lo que incluye caracteres alfanuméricos y guiones bajos. Esta expresión habitual hace coincidir los archivos de registro con y sin marcas temporales en su nombre. En este ejemplo, el administrador de registro carga los siguientes archivos de registro:

- `hello_world.log`: el archivo de registro más reciente de la aplicación Hello World.
- `hello_world_2020_12_15_17_0.log`: un archivo de registro más antiguo de la aplicación Hello World.

Predeterminado:*`componentName`*`\\\\w*.log`, donde *`componentName`* es el nombre del componente para esta configuración de registro.

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en CloudWatch Logs.

Valor predeterminado: `false`

`multiLineStartPattern`

(Opcional) Expresión habitual que identifica cuándo un mensaje de registro de una línea nueva es un mensaje de registro nuevo. Si la expresión regular no coincide

con la nueva línea, el componente administrador de registros anexa la nueva línea al mensaje de registro de la línea anterior.

De forma predeterminada, el componente administrador de registros comprueba si la línea comienza con un carácter de espacio en blanco, como un tabulador o un espacio. Si no es así, el administrador de registros trata esa línea como un nuevo mensaje de registro. De lo contrario, anexa esa línea al mensaje de registro actual. Este comportamiento garantiza que el componente administrador de registros no divida los mensajes que ocupan varias líneas, como los rastreos de pila.

`periodicUploadIntervalSec`

(Opcional) El periodo en segundos durante el que el componente administrador de registros comprueba si hay nuevos archivos de registro para cargar.

Valor predeterminado: `300` (5 minutos)

Mínimo: `0.000001` (1 microsegundo)

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica cargar los registros del sistema y los registros de los `com.example.HelloWorld` componentes en CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
```

```
  "periodicUploadIntervalSec": "300"
}
```

Example Ejemplo: configuración para cargar varios archivos de registro activos mediante la versión 2.3.1 del administrador de registros

El siguiente ejemplo de configuración es el ejemplo recomendado si desea dirigirse a varios archivos de registro activos. En este ejemplo de configuración se especifican los archivos de registro activos en los que desea cargarlos CloudWatch. Si utiliza este ejemplo de configuración, la configuración también cargará todos los archivos rotados que coincidan con `logFileRegex`. Este ejemplo de configuración es compatible con el administrador de registros versión 2.3.1.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.2.x

`logsUploaderConfiguration`

(Opcional) La configuración de los registros que carga el componente administrador de registros. Este objeto contiene la siguiente información:

`systemLogsConfiguration`

(Opcional) La configuración de los registros del sistema del software AWS IoT Greengrass principal, que incluyen los registros del núcleo y los componentes del complemento de Greengrass. Especifique esta configuración para permitir que el componente administrador de registros administre los registros del sistema. Este objeto contiene la siguiente información:

`uploadToCloudWatch`

(Opcional) Puede cargar los registros del sistema en CloudWatch Logs.

Valor predeterminado: `false`

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si configura el componente núcleo de Greengrass para generar registros en formato JSON. Para habilitar los registros en formato JSON, especifique `JSON` para el parámetro de formato de registro (`logging.format`).

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de los archivos de registro del sistema Greengrass en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro del sistema Greengrass supera este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro del sistema Greengrass más antiguos.

Este parámetro equivale al parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño máximo total del registro del sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes

- `GB`: gigabytes

Valor predeterminado: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`componentLogsConfigurationMap`

(Opcional) Un mapa de las configuraciones de registro de los componentes del dispositivo principal. Cada objeto `componentName` de este mapa define la configuración de registro del componente o la aplicación. El componente gestor de registros carga estos registros de componentes en Logs. CloudWatch

> **Note**
>
> Si está actualizando desde una versión del administrador de registros anterior a la versión 2.2.0, puede seguir utilizando la lista `componentLogsConfiguration` en lugar de `componentLogsConfigurationMap`. No obstante, le recomendamos que utilice el formato de mapa para poder utilizar actualizaciones de combinación y restablecimiento para modificar configuraciones de componentes específicos. Para obtener información sobre el parámetro `componentLogsConfiguration`, consulte los parámetros de configuración de la versión 2.1.x de este componente.

`componentName`

La configuración de registro del componente o la aplicación *`componentName`* de esta configuración de registro. Puede especificar el nombre de un componente de Greengrass u otro valor para identificar este grupo de registro.

Cada objeto contiene la siguiente información:

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si los registros de este componente utilizan un formato JSON específico, que puede encontrar en el repositorio del módulo de AWS IoT Greengrass registro de. GitHub

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de todos los archivos de registro de este componente en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro de este componente supere este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de este componente.

Este parámetro está relacionado con el parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para este componente.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`logFileDirectoryPath`

(Opcional) La ruta a la carpeta que contiene los archivos de registro de este componente.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Predeterminado: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Opcional) Expresión habitual que especifica el formato de nombre del archivo de registro que utiliza el componente o la aplicación. El componente administrador de registro utiliza esta expresión habitual para identificar los archivos de registro de la carpeta situada en `logFileDirectoryPath`.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Si su componente o aplicación rota los archivos de registro, especifique una expresión habitual que coincida con los nombres de los archivos de registro rotados. Por ejemplo, puede especificar **`hello_world\\\\w*.log`** para cargar los registros de una aplicación de Hello World. El patrón `\\\\w*` coincide con cero o más caracteres de palabra, lo que incluye caracteres alfanuméricos y guiones bajos. Esta expresión habitual hace coincidir los archivos de registro con y sin marcas temporales en su nombre. En este ejemplo, el administrador de registro carga los siguientes archivos de registro:

- `hello_world.log`: el archivo de registro más reciente de la aplicación Hello World.
- `hello_world_2020_12_15_17_0.log`: un archivo de registro más antiguo de la aplicación Hello World.

Predeterminado:*`componentName`*`\\\\w*.log`, donde *`componentName`* es el nombre del componente para esta configuración de registro.

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en CloudWatch Logs.

Valor predeterminado: `false`

`multiLineStartPattern`

(Opcional) Expresión habitual que identifica cuándo un mensaje de registro de una línea nueva es un mensaje de registro nuevo. Si la expresión regular no coincide con la nueva línea, el componente administrador de registros anexa la nueva línea al mensaje de registro de la línea anterior.

De forma predeterminada, el componente administrador de registros comprueba si la línea comienza con un carácter de espacio en blanco, como un tabulador o un espacio. Si no es así, el administrador de registros trata esa línea como un nuevo mensaje de registro. De lo contrario, anexa esa línea al mensaje de registro actual. Este comportamiento garantiza que el componente administrador de registros no divida los mensajes que ocupan varias líneas, como los rastreos de pila.

`periodicUploadIntervalSec`

(Opcional) El periodo en segundos durante el que el componente administrador de registros comprueba si hay nuevos archivos de registro para cargar.

Valor predeterminado: `300` (5 minutos)

Mínimo: `0.000001` (1 microsegundo)

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica cargar los registros del sistema y los registros de los `com.example.HelloWorld` componentes en CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

`logsUploaderConfiguration`

(Opcional) La configuración de los registros que carga el componente administrador de registros. Este objeto contiene la siguiente información:

`systemLogsConfiguration`

(Opcional) La configuración de los registros del sistema del software AWS IoT Greengrass principal, que incluyen los registros del núcleo y los componentes del complemento de Greengrass. Especifique esta configuración para permitir que el componente administrador de registros administre los registros del sistema. Este objeto contiene la siguiente información:

`uploadToCloudWatch`

(Opcional) Puede cargar los registros del sistema en CloudWatch Logs.

Valor predeterminado: `false`

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si configura el componente núcleo de Greengrass para generar registros en formato JSON. Para habilitar los registros en formato JSON, especifique `JSON` para el parámetro de formato de registro (`logging.format`).

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de los archivos de registro del sistema Greengrass en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro del sistema Greengrass supera este tamaño total máximo,

el software AWS IoT Greengrass Core elimina los archivos de registro del sistema Greengrass más antiguos.

Este parámetro equivale al parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño máximo total del registro del sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`componentLogsConfiguration`

(Opcional) Una lista de las configuraciones de registro de los componentes del dispositivo principal. Cada configuración de esta lista define la configuración de registro de un componente o aplicación. El componente gestor de registros carga estos registros de componentes en Logs CloudWatch

Cada objeto contiene la siguiente información:

`componentName`

El nombre del componente o la aplicación para esta configuración de registro. Puede especificar el nombre de un componente de Greengrass u otro valor para identificar este grupo de registro.

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si los registros de este componente utilizan un formato JSON

específico, que se encuentra en el repositorio del módulo de AWS IoT Greengrass registro. GitHub

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de todos los archivos de registro de este componente en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro de este componente supere este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de este componente.

Este parámetro está relacionado con el parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para este componente.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`logFileDirectoryPath`

(Opcional) La ruta a la carpeta que contiene los archivos de registro de este componente.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Predeterminado: `/greengrass/v2/logs`.

`logFileRegex`

(Opcional) Expresión habitual que especifica el formato de nombre del archivo de registro que utiliza el componente o la aplicación. El componente administrador de registro utiliza esta expresión habitual para identificar los archivos de registro de la carpeta situada en `logFileDirectoryPath`.

No es necesario especificar este parámetro para los componentes de Greengrass que imprimen en salida estándar (stdout) y error estándar (stderr).

Si su componente o aplicación rota los archivos de registro, especifique una expresión habitual que coincida con los nombres de los archivos de registro rotados. Por ejemplo, puede especificar **`hello_world\\\\w*.log`** para cargar los registros de una aplicación de Hello World. El patrón `\\\\w*` coincide con cero o más caracteres de palabra, lo que incluye caracteres alfanuméricos y guiones bajos. Esta expresión habitual hace coincidir los archivos de registro con y sin marcas temporales en su nombre. En este ejemplo, el administrador de registro carga los siguientes archivos de registro:

- `hello_world.log`: el archivo de registro más reciente de la aplicación Hello World.
- `hello_world_2020_12_15_17_0.log`: un archivo de registro más antiguo de la aplicación Hello World.

Predeterminado:*componentName*`\\\\w*.log`, donde *componentName* es el nombre del componente para esta configuración de registro.

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en CloudWatch Logs.

Valor predeterminado: `false`

`multiLineStartPattern`

(Opcional) Expresión habitual que identifica cuándo un mensaje de registro de una línea nueva es un mensaje de registro nuevo. Si la expresión regular no coincide con la

nueva línea, el componente administrador de registros anexa la nueva línea al mensaje de registro de la línea anterior.

De forma predeterminada, el componente administrador de registros comprueba si la línea comienza con un carácter de espacio en blanco, como un tabulador o un espacio. Si no es así, el administrador de registros trata esa línea como un nuevo mensaje de registro. De lo contrario, anexa esa línea al mensaje de registro actual. Este comportamiento garantiza que el componente administrador de registros no divida los mensajes que ocupan varias líneas, como los rastreos de pila.

`periodicUploadIntervalSec`

(Opcional) El periodo en segundos durante el que el componente administrador de registros comprueba si hay nuevos archivos de registro para cargar.

Valor predeterminado: `300` (5 minutos)

Mínimo: `0.000001` (1 microsegundo)

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica cargar los registros del sistema y los registros de los `com.example.HelloWorld` componentes en CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
```

```
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.0.x

`logsUploaderConfiguration`

(Opcional) La configuración de los registros que carga el componente administrador de registros. Este objeto contiene la siguiente información:

`systemLogsConfiguration`

(Opcional) La configuración de los registros del sistema del software AWS IoT Greengrass principal. Especifique esta configuración para permitir que el componente administrador de registros administre los registros del sistema. Este objeto contiene la siguiente información:

`uploadToCloudWatch`

(Opcional) Puede cargar los registros del sistema en CloudWatch Logs.

Valor predeterminado: `false`

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si configura el componente núcleo de Greengrass para generar registros en formato JSON. Para habilitar los registros en formato JSON, especifique `JSON` para el parámetro de formato de registro (`logging.format`).

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de los archivos de registro del sistema Greengrass en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro del sistema Greengrass supera este tamaño total máximo,

el software AWS IoT Greengrass Core elimina los archivos de registro del sistema Greengrass más antiguos.

Este parámetro equivale al parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño máximo total del registro del sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`componentLogsConfiguration`

(Opcional) Una lista de las configuraciones de registro de los componentes del dispositivo principal. Cada configuración de esta lista define la configuración de registro de un componente o aplicación. El componente gestor de registros carga estos registros de componentes en Logs CloudWatch

Cada objeto contiene la siguiente información:

`componentName`

El nombre del componente o la aplicación para esta configuración de registro. Puede especificar el nombre de un componente de Greengrass u otro valor para identificar este grupo de registro.

`minimumLogLevel`

(Opcional) El nivel mínimo de mensajes de registro que se deben cargar. Este nivel mínimo se aplica solo si los registros de este componente utilizan un formato JSON

específico, que se encuentra en el repositorio del módulo de AWS IoT Greengrass registro. GitHub

Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`diskSpaceLimit`

(Opcional) El tamaño total máximo de todos los archivos de registro de este componente en la unidad en la que especifique `diskSpaceLimitUnit`. Cuando el tamaño total de los archivos de registro de este componente supere este tamaño total máximo, el software AWS IoT Greengrass Core elimina los archivos de registro más antiguos de este componente.

Este parámetro está relacionado con el parámetro de límite de tamaño de registro (`totalLogsSizeKB`) del componente núcleo de Greengrass. El software AWS IoT Greengrass Core utiliza el mínimo de los dos valores como tamaño de registro total máximo para este componente.

`diskSpaceLimitUnit`

(Opcional) La unidad para `diskSpaceLimit`. Puede elegir entre las siguientes opciones:

- `KB`: kilobytes
- `MB`: megabytes
- `GB`: gigabytes

Valor predeterminado: `KB`

`logFileDirectoryPath`

La ruta a la carpeta que contiene los archivos de registro de este componente.

Para cargar los registros de un componente de Greengrass, especifique **/*greengrass/v2*/logs** y remplace */greengrass/v2* por su carpeta raíz de Greengrass.

`logFileRegex`

Expresión habitual que especifica el formato de nombre del archivo de registro que utiliza el componente o la aplicación. El componente administrador de registro utiliza esta expresión habitual para identificar los archivos de registro de la carpeta situada en `logFileDirectoryPath`.

Para cargar los registros de un componente de Greengrass, especifique una expresión regular que coincida con los nombres de los archivos de registro rotados. Por ejemplo, puede especificar **`com.example.HelloWorld\\w*.log`** para cargar los registros de un componente Hello World. El patrón `\\w*` coincide con cero o más caracteres de palabra, lo que incluye caracteres alfanuméricos y guiones bajos. Esta expresión habitual hace coincidir los archivos de registro con y sin marcas temporales en su nombre. En este ejemplo, el administrador de registro carga los siguientes archivos de registro:

- `com.example.HelloWorld.log`: el archivo de registro más reciente del componente Hello World.
- `com.example.HelloWorld_2020_12_15_17_0.log`: un archivo de registro más antiguo del componente Hello World. El núcleo de Greengrass agrega una marca temporal rotativa a los archivos de registro.

`deleteLogFileAfterCloudUpload`

(Opcional) Puede eliminar un archivo de registro después de que el componente del administrador de registros cargue los registros en Logs. CloudWatch

Valor predeterminado: `false`

`multiLineStartPattern`

(Opcional) Expresión habitual que identifica cuándo un mensaje de registro de una línea nueva es un mensaje de registro nuevo. Si la expresión regular no coincide con la nueva línea, el componente administrador de registros anexa la nueva línea al mensaje de registro de la línea anterior.

De forma predeterminada, el componente administrador de registros comprueba si la línea comienza con un carácter de espacio en blanco, como un tabulador o un

espacio. Si no es así, el administrador de registros trata esa línea como un nuevo mensaje de registro. De lo contrario, anexa esa línea al mensaje de registro actual. Este comportamiento garantiza que el componente administrador de registros no divida los mensajes que ocupan varias líneas, como los rastreos de pila.

`periodicUploadIntervalSec`

(Opcional) El periodo en segundos durante el que el componente administrador de registros comprueba si hay nuevos archivos de registro para cargar.

Valor predeterminado: `300` (5 minutos)

Mínimo: `0.000001` (1 microsegundo)

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica cargar los registros del sistema y los registros de los `com.example.HelloWorld` componentes en CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

De uso

El componente administrador de registros se carga en los siguientes grupos de registro y flujos de registro.

2.1.0 and later

Nombre del grupo de registro

```
/aws/greengrass/componentType/region/componentName
```

El nombre del grupo de registro utiliza las siguientes variables:

- `componentType`: el tipo del componente, que puede ser uno de los siguientes:
  - `GreengrassSystemComponent`: este grupo de registro incluye los registros de los componentes del núcleo y del complemento, que se ejecutan en la misma JVM que el núcleo de Greengrass. El componente forma parte del núcleo de Greengrass.
  - `UserComponent`: este grupo de registro incluye registros de componentes genéricos, componentes de Lambda y otras aplicaciones del dispositivo. El componente no forma parte del núcleo de Greengrass.

  Para obtener más información, consulte Tipos de componentes.
- `region`— La AWS región que utiliza el dispositivo principal.
- `componentName`: el nombre del componente. En el caso de los registros del sistema, este valor es `System`.

Nombre del flujo de registro

```
/date/thing/thingName
```

El nombre del flujo de registro utiliza las siguientes variables:

- `date`: la fecha del registro, por ejemplo, `2020/12/15`. El componente administrador de registros usa el formato `yyyy/MM/dd`.
- `thingName`: el nombre del dispositivo principal.

> **Note**
>
> Si el nombre de un objeto contiene dos puntos (`:`), el administrador de registros los sustituye por un signo más (`+`).

2.0.x

Nombre del grupo de registro

```
/aws/greengrass/componentType/region/componentName
```

El nombre del grupo de registro utiliza las siguientes variables:

- `componentType`: el tipo del componente, que puede ser uno de los siguientes:
  - `GreengrassSystemComponent`: el componente forma parte del núcleo de Greengrass.
  - `UserComponent`: el componente no forma parte del núcleo de Greengrass. El administrador de registros usa este tipo para los componentes de Greengrass y otras aplicaciones del dispositivo.
- `region`— La AWS región que utiliza el dispositivo principal.
- `componentName`: el nombre del componente. En el caso de los registros del sistema, este valor es `System`.

Nombre del flujo de registro

```
/date/deploymentTargets/thingName
```

El nombre del flujo de registro utiliza las siguientes variables:

- `date`: la fecha del registro, por ejemplo, `2020/12/15`. El componente administrador de registros usa el formato `yyyy/MM/dd`.
- `deploymentTargets`: los objetos cuyas implementaciones incluyen el componente. El componente administrador de registros separa cada objetivo mediante una barra oblicua. Si el componente se ejecuta en el dispositivo principal como resultado de una implementación local, este valor es `LOCAL_DEPLOYMENT`.

  Considere un ejemplo en el que tiene un dispositivo principal con nombre `MyGreengrassCore` y el dispositivo principal tiene dos implementaciones:

- Una implementación dirigida al dispositivo principal, `MyGreengrassCore`.
- Una implementación dirigida a un grupo de objetos con nombre `MyGreengrassCoreGroup`, que contiene el dispositivo principal.

Los `deploymentTargets` de este dispositivo principal son `thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup`.

- `thingName`: el nombre del dispositivo principal.

Formatos para las entradas de registro.

El núcleo de Greengrass escribe los archivos de registro en formato de cadena o JSON. En el caso de los registros del sistema, usted controla el formato configurando el campo `format` de la entrada `logging`. Puede encontrar la entrada `logging` en el archivo de configuración del componente núcleo de Greengrass. Para obtener más información, consulte la Configuración del núcleo de Greengrass.

El formato de texto es de formato libre y acepta cualquier cadena. El siguiente mensaje del servicio de estado de la flota es un ejemplo de registro con formato de cadena:

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

Debe usar el formato JSON si quiere ver los registros con el comando registros de la CLI de Greengrass o interactuar con los registros mediante programación. En el siguiente ejemplo, se describe la forma JSON:

```
{
   "loggerName": <string>,
   "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
   "eventType": <string, optional>,
   "cause": <string, optional>,
   "contexts": {},
   "thread": <string>,
   "message": <string>,
   "timestamp": <epoch time> # Needs to be epoch time
}
```

Para controlar la salida de los registros de su componente, puede usar la opción de configuración `minimumLogLevel`. Para usar esta opción, el componente debe escribir sus entradas de registro en formato JSON. Debe usar el mismo formato que el archivo de registro del sistema.

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.3.11	Mejoras y correcciones de errores • Soluciona un problema por el que la configuración del tiempo de ejecución de Log Manager aumentaba indefinidamente debido a que la información de los archivos de registro cargados era obsoleta.
2.3.10	Nuevas características • Agrega una nueva clave de configuración (`updateToTlogIntervalSec`) para controlar la frecuencia con la que los detalles de los eventos de carga de registros se conservan en el registro de transacciones local (`config.tlog`). Mejoras y correcciones de errores • Mejora el administrador de registros para actualizar el cliente de CloudWatch en caso de errores en la conexión del socket.
2.3.9	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.3.8	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.3.7	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.3.6	Mejoras y correcciones de errores • Ajusta los niveles de registro para detectar determinados errores.
2.3.5	Mejoras Mejora la velocidad de carga de los registros. Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.3.4	Mejoras y correcciones de errores • Suma soporte para configurar el parámetro `periodicUploadIntervalSec` en valores fraccionarios. El mínimo es 1 microsegundo.

Versión	Cambios
	• Soluciona un problema por el que el administrador de registros no respetaba los límites. CloudWatch `putLogEvents`
2.3.3	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.3.2	Mejoras y correcciones de errores • Mejora de la administración del espacio para que los archivos de registro no se eliminen antes de cargarlos. • Solución de problemas relacionados con la administración de la memoria caché. • Correcciones de errores y mejoras menores adicionales.
2.3.1	Mejoras y correcciones de errores • Soluciona un problema por el que los grupos de archivos de destino con varios archivos de registro activos cargaban entradas duplicadas en ellas. CloudWatch • Correcciones de errores y mejoras menores adicionales.
2.3.0	**Note** Recomendamos que actualice el núcleo de Greengrass a 2.9.1 cuando actualice el administrador de registros a 2.3.0. Nuevas características Reducción de las demoras en el registro cuando se procesan y cargan directamente los archivos de registro activos en lugar de esperar a que se roten los archivos nuevos. Mejoras y correcciones de errores • Mejora de la compatibilidad con la rotación de registros cuando se rotan archivos con un nombre único. • Correcciones de errores y mejoras menores adicionales.

Versión	Cambios
2.2.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.2.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.2.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.2.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.2.4	Mejoras y correcciones de errores • Mejora de la estabilidad cuando se gestionan configuraciones no válidas. • Correcciones y mejoras menores adicionales.
2.2.3	Mejoras y correcciones de errores • Mejora de la estabilidad en ciertos escenarios en los que el componente se reinicia o encuentra errores. • Solución de problemas por los que los mensajes de registro y los archivos de registro de gran tamaño no se cargaban en determinadas situaciones. • Soluciona problemas relacionados con la forma en que este component e administra las actualizaciones de restablecimiento de la configuración. • Solución de un problema por el que un valor de configuración `null` `diskSpaceLimit` impedía la implementación del componente.
2.2.2	Mejoras y correcciones de errores • Suma compatibilidad con los mensajes de registro de más de 256 kilobytes. El componente administrador de registros divide estos mensajes de registro de gran tamaño en varios mensajes con la misma marca temporal de evento de registro.

Versión	Cambios
2.2.1	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.2.0	Nueva característica • Agrega el parámetro de configuración `componentLogsConfigurationMap` para admitir un formato de mapa para las configuraciones de registro de componentes. Cada objeto `componentName` del mapa define la configuración de registro de un componente o una aplicación.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Mejoras y correcciones de errores • Solución de un problema por el que la configuración del registro del sistema no se actualizaba en algunos casos.
2.1.0	Mejoras y correcciones de errores • Utilice valores predeterminados para `logFileDirectoryPath` y `logFileRegex` que funcionen para los componentes de Greengrass que imprimen con salida estándar (stdout) y error estándar (stderr). • Dirija correctamente el tráfico a través de un proxy de red configurado al cargar registros en Logs. CloudWatch • Maneje correctamente los dos puntos (`:`) en los nombres de los flujos de registro. CloudWatch Los nombres de los flujos de registro de registros no admiten signos de dos puntos. • Simplifique los nombres del flujo de registro al eliminar los nombres de los grupos de objetos del flujo de registro. • Elimine un mensaje de registro de errores que se imprime con un comportamiento normal.

Versión	Cambios
2.0.x	Versión inicial.

Componentes de machine learning

AWS IoT Greengrass proporciona los siguientes componentes de aprendizaje automático que puede implementar en dispositivos compatibles para realizar inferencias de aprendizaje automático mediante modelos entrenados en Amazon SageMaker AI o con sus propios modelos previamente entrenados que se almacenan en Amazon S3.

AWS proporciona las siguientes categorías de componentes de aprendizaje automático:

- Componente de modelo: contiene modelos de machine learning como artefactos de Greengrass.
- Componente de tiempo de ejecución: contiene el script que instala el marco de machine learning y sus dependencias en el dispositivo principal de Greengrass.
- Componente de inferencia: contiene el código de inferencia e incluye las dependencias de los componentes para instalar el marco de machine learning y descargar modelos de machine learning previamente entrenados.

Puede utilizar el código de inferencia de muestra y los modelos previamente entrenados de los componentes AWS de aprendizaje automático proporcionados para realizar la clasificación de imágenes y la detección de objetos mediante DLR y Lite. TensorFlow Para realizar inferencias de machine learning personalizadas con sus propios modelos almacenados en Amazon S3 o, para utilizar un marco de machine learning diferente, puede utilizar las recetas de estos componentes públicos como plantillas para crear componentes de machine learning personalizados. Para obtener más información, consulte Personalización de sus componentes de machine learning.

AWS IoT Greengrass también incluye un componente AWS proporcionado para gestionar la instalación y el ciclo de vida del agente SageMaker AI Edge Manager en los dispositivos principales de Greengrass. Con SageMaker AI Edge Manager, puede utilizar los modelos compilados por Amazon SageMaker AI NEO directamente en su dispositivo principal. Para obtener más información, consulte Utilice Amazon SageMaker AI Edge Manager en los dispositivos principales de Greengrass.

En la siguiente tabla se enumeran los componentes de aprendizaje automático que están disponibles en AWS IoT Greengrass.

> **Note**
>
> Varios AWS de los componentes proporcionados dependen de versiones secundarias específicas del núcleo de Greengrass. Debido a esta dependencia, es necesario actualizar estos componentes al actualizar el núcleo de Greengrass a una nueva versión secundaria. Para obtener información sobre las versiones específicas del núcleo de las que depende cada componente, consulte el tema del componente correspondiente. Para más información sobre la actualización del núcleo, consulte Actualización del software AWS IoT Greengrass Core (OTA).

Cuando un componente tiene un tipo de componente genérico y de Lambda, la versión actual del componente es del tipo genérico y la versión anterior del componente es del tipo de Lambda.

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
SageMaker Administrador AI Edge	Implementa el agente Amazon SageMaker AI Edge Manager en el dispositivo principal de Greengrass.	Genérico	Linux, Windows	No
Clasificación de imágenes de DLR	Componente de inferencia que utiliza el almacén de modelos de clasificación de imágenes de DLR y el componente de tiempo	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
	de ejecución de DLR como dependencias para instalar el DLR, descargar modelos de clasificación de imágenes de muestra y realizar inferencias de clasificación de imágenes en los dispositivos compatibles.			

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
Detección de objetos del DLR	Componente de inferencia que utiliza el almacén de modelos de detección de objetos del DLR y el componente de tiempo de ejecución del DLR como dependencias para instalar el DLR, descargar modelos de detección de objetos de muestra y realizar inferencias de detección de objetos en los dispositivos compatibles.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
Almacén de modelos de clasificación de imágenes de DLR	Componente de modelo que contiene ejemplos de ResNet -50 modelos de clasificación de imágenes como artefactos de Greengrass.	Genérico	Linux, Windows	No
Almacén de modelos de detección de objetos del DLR	Componente de modelo que contiene ejemplos de modelos de detección de YOLOv3 objetos como artefactos de Greengrass.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
Tiempo de ejecución de DLR	Componente de tiempo de ejecución que contiene una cadena de instalación que se utiliza para instalar el DLR y sus dependencias en el dispositivo principal de Greengrass.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Clasificación de imágenes Lite	Componente de inferencia que utiliza el TensorFlow almacén de modelos de clasificación de imágenes de TensorFlow Lite y el componente de tiempo de ejecución de Lite como dependencias para instalar TensorFlow Lite, descargar modelos de clasificación de imágenes de muestra y realizar inferencias de clasificación de imágenes en dispositivos compatibles.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Detección de objetos Lite	Componente de inferencia que utiliza la tienda de modelos de detección de objetos de TensorFlow Lite y el componente de tiempo de ejecución de TensorFlow Lite como dependencias para instalar TensorFlow Lite, descargar modelos de detección de objetos de muestra y realizar inferencias de detección de objetos en dispositivos compatibles.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Tienda de modelos de clasificación de imágenes Lite	Componente de modelo que contiene un modelo MobileNet v1 de muestra como artefacto de Greengrass.	Genérico	Linux, Windows	No
TensorFlow Tienda de modelos de detección de objetos Lite	Componente de modelo que contiene un MobileNet modelo de muestra de detección de disparo único (SSD) como un artefacto de Greengrass.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Tiempo de ejecución Lite	Componente de tiempo de ejecución que contiene un script de instalación que se utiliza para instalar TensorFlow Lite y sus dependencias en el dispositivo principal de Greengrass.	Genérico	Linux, Windows	No

SageMaker Administrador AI Edge

> **Important**
>
> SageMaker AI Edge Manager se suspendió el 26 de abril de 2024. Para obtener más información sobre cómo seguir implementando sus modelos en dispositivos periféricos, consulte el final del ciclo de vida de SageMaker AI Edge Manager.

El componente Amazon SageMaker AI Edge Manager (`aws.greengrass.SageMakerEdgeManager`) instala el binario del agente SageMaker AI Edge Manager.

SageMaker AI Edge Manager proporciona una gestión de modelos para dispositivos periféricos, de forma que pueda optimizar, proteger, supervisar y mantener los modelos de aprendizaje automático en flotas de dispositivos periféricos. El componente SageMaker AI Edge Manager instala y gestiona el ciclo de vida del agente SageMaker AI Edge Manager en su dispositivo principal. También puede usar SageMaker AI Edge Manager para empaquetar y usar modelos compilados por SageMaker AI

NEO como componentes de modelos en los dispositivos principales de Greengrass. Para obtener más información sobre el uso del agente SageMaker AI Edge Manager en su dispositivo principal, consulte. Utilice Amazon SageMaker AI Edge Manager en los dispositivos principales de Greengrass

SageMaker El componente AI Edge Manager v1.3.x instala el agente binario de Edge Manager v1.20220822.836f3023. Para obtener más información sobre las versiones binarias del agente Administrador de periféricos, consulte Agent administrador de periféricos.

> **Note**
>
> El componente AI SageMaker Edge Manager solo está disponible en los siguientes casos Regiones de AWS:
>
> - Este de EE. UU. (Ohio)
> - Este de EE. UU. (Norte de Virginia)
> - Oeste de EE. UU. (Oregón)
> - UE (Fráncfort)
> - UE (Irlanda)
> - Asia-Pacífico (Tokio)

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 1.3.x

- 1.2.x
- 1.1.x
- 1.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- Un dispositivo principal de Greengrass que se ejecuta en Amazon Linux 2, una plataforma de Linux basada en Debian (x86_64 o Armv8) o Windows (x86_64). Si no dispone de una, consulte Tutorial: Introducción a AWS IoT Greengrass V2.
- Python 3.6 o posterior, incluido `pip` para la versión de Python, instalado en el dispositivo principal.
- El rol del dispositivo de Greengrass se configuró con lo siguiente:
  - Una relación de confianza que permite a `credentials.iot.amazonaws.com` y a `sagemaker.amazonaws.com` asumir el rol, como se muestra en el siguiente ejemplo de política de IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
```

```
        },
        "Action": "sts:AssumeRole"
      },
      {
        "Effect": "Allow",
        "Principal": {
          "Service": "sagemaker.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
      }
    ]
}
```

- La política gestionada por AmazonSageMakerEdgeDeviceFleetPolicyIAM.
- La acción `s3:PutObject`, como se muestra en el siguiente ejemplo de política de IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Un bucket de Amazon S3 creado en el mismo dispositivo principal de Greengrass Cuenta de AWS y Región de AWS en el mismo que él. SageMaker AI Edge Manager requiere un depósito S3 para crear una flota de dispositivos perimetrales y almacenar datos de muestra derivados de la ejecución de inferencias en su dispositivo. Para más información sobre la creación de buckets de S3, consulte Introducción a Amazon S3.
- Una flota de dispositivos periféricos de SageMaker IA que utiliza el mismo alias de AWS IoT rol que su dispositivo principal de Greengrass. Para obtener más información, consulte Creación de una flota de dispositivos de periferia.
- Su dispositivo principal de Greengrass registrado como dispositivo perimetral en su flota de dispositivos SageMaker AI Edge. El nombre del dispositivo perimetral debe coincidir con el

AWS IoT nombre del dispositivo principal. Para obtener más información, consulte Registro del dispositivo principal de Greengrass.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`edge.sagemaker.` *`region`*`.amazonaws.com`	443	Sí	Comprueba el estado de registro del dispositivo y envía las métricas a SageMaker AI.
`*.s3.amazonaws.com`	443	Sí	Cargue los datos de captura en el bucket de S3 que especifique. Puede sustituirlos por * con el nombre de cada

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			bucket en el que publique los datos.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

1.3.5 and 1.3.6

En la siguiente tabla, se muestran las dependencias de las versiones 1.3.5 y 1.3.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.3.4

En la siguiente tabla, se muestran las dependencias de la versión 1.3.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.3.3

En la siguiente tabla, se muestran las dependencias de la versión 1.3.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.3.2

En la siguiente tabla, se muestran las dependencias de la versión 1.3.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.3.1

En la siguiente tabla, se muestran las dependencias de la versión 1.3.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=0.0.0	Rígido

1.1.1 - 1.3.0

En la siguiente tabla, se muestran las dependencias de las versiones 1.1.1 a 1.3.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.1.0

En la siguiente tabla, se muestran las dependencias de la versión 1.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.0.3

En la siguiente tabla, se muestran las dependencias de la versión 1.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=0.0.0	Rígido

1.0.1 and 1.0.2

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.1 y 1.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

1.0.0

En la siguiente tabla, se muestran las dependencias de la versión 1.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

> **Note**
>
> En esta sección, se describen los parámetros de configuración que se establecen en el componente. Para obtener más información sobre la configuración de SageMaker AI Edge Manager correspondiente, consulte Edge Manager Agent en la Guía para desarrolladores de Amazon SageMaker AI.

`DeviceFleetName`

El nombre de la flota de dispositivos SageMaker AI Edge Manager que contiene su dispositivo principal Greengrass.

Al implementar este componente, debe especificar un valor para este parámetro en la actualización de configuración.

`BucketName`

El nombre del bucket de S3 en donde se cargan los datos de inferencia capturados. El nombre del bucket debe contener la cadena `sagemaker`.

Si se establece `CaptureDataDestination` en `Cloud`, o si se establece `CaptureDataPeriodicUpload` en `true`, debe especificar un valor para este parámetro en la actualización de la configuración al implementar este componente.

> **Note**
>
> La captura de datos es una función de SageMaker IA que se utiliza para cargar entradas de inferencia, resultados de inferencias y datos de inferencia adicionales a un bucket de S3 o a un directorio local para futuros análisis. Para obtener más información sobre el uso de datos de captura con SageMaker AI Edge Manager, consulte Manage Model en la Guía para desarrolladores de Amazon SageMaker AI.

`CaptureDataBatchSize`

(Opcional) El tamaño de un lote de solicitudes de datos de captura que gestiona el agente. Este valor debe ser menor que el tamaño de búfer que especifique en `CaptureDataBufferSize`. Le recomendamos que no exceda la mitad del tamaño del búfer.

El agente gestiona un lote de solicitudes cuando el número de solicitudes del búfer es igual al `CaptureDataBatchSize` número o cuando `CaptureDataPushPeriodSeconds` transcurre el intervalo, lo que ocurra primero.

Valor predeterminado: `10`

`CaptureDataBufferSize`

(Opcional) El número máximo de solicitudes de datos de captura almacenadas en el búfer.

Valor predeterminado: `30`

`CaptureDataDestination`

(Opcional) El destino en el que se almacenan los datos capturados. Este parámetro puede tener uno de los siguientes valores:

- `Cloud`: carga los datos capturados en el bucket de S3 que especifique en `BucketName`.
- `Disk`: escribe los datos capturados en el directorio de trabajo del componente.

Si especifica `Disk`, también puede optar por cargar periódicamente los datos capturados en su bucket de S3 configurando `CaptureDataPeriodicUpload` en `true`.

Valor predeterminado: `Cloud`

`CaptureDataPeriodicUpload`

(Opcional) Valor de cadena que especifica si se deben cargar periódicamente los datos capturados. Los valores admitidos son `true` y `false`.

Establezca este parámetro en `true` si ha establecido `CaptureDataDestination` en `Disk` y si también desea que el agente cargue periódicamente los datos capturados en su bucket de S3.

Valor predeterminado: `false`

`CaptureDataPeriodicUploadPeriodSeconds`

(Opcional) El intervalo en segundos en el que el agente de SageMaker AI Edge Manager carga los datos capturados en el depósito de S3. Si usa este parámetro, debe establecer `CaptureDataPeriodicUpload` en `true`.

Valor predeterminado: 8

`CaptureDataPushPeriodSeconds`

(Opcional) El intervalo en segundos en el que el agente de SageMaker AI Edge Manager gestiona un lote de solicitudes de datos de captura desde el búfer.

El agente gestiona un lote de solicitudes cuando el número de solicitudes del búfer es igual al `CaptureDataBatchSize` número o cuando `CaptureDataPushPeriodSeconds` transcurre el intervalo, lo que ocurra primero.

Valor predeterminado: `4`

`CaptureDataBase64EmbedLimit`

(Opcional) El tamaño máximo en bytes de los datos capturados que carga el agente de SageMaker AI Edge Manager.

Valor predeterminado: `3072`

`FolderPrefix`

(Opcional) El nombre de la carpeta en la que el agente escribe los datos capturados. Si se establece `CaptureDataDestination` en `Disk`, el agente crea la carpeta en el directorio especificado por `CaptureDataDiskPath`. Si se establece `CaptureDataDestination` en `Cloud`, o si se establece `CaptureDataPeriodicUpload` en `true`, el agente crea la carpeta en el bucket de S3.

Valor predeterminado: `sme-capture`

`CaptureDataDiskPath`

Esta función está disponible en la versión 1.1.0 y en las versiones posteriores del componente SageMaker AI Edge Manager.

(Opcional) La ruta a la carpeta en la que el agente crea la carpeta de datos capturados. Si establece `CaptureDataDestination` en `Disk`, el agente crea la carpeta de datos capturados en este directorio. Si no especifica este valor, el agente crea la carpeta de datos capturados en el directorio de trabajo del componente. Utilice el parámetro `FolderPrefix` para especificar el nombre de la carpeta de datos capturados.

Valor predeterminado: *`/greengrass/v2`*`/work/aws.greengrass.SageMakerEdgeManager/capture`

`LocalDataRootPath`

Esta función está disponible en la versión 1.2.0 y en las versiones posteriores del componente SageMaker AI Edge Manager.

(Opcional) La ruta en la que este componente almacena los siguientes datos en el dispositivo principal:

- La base de datos local para los datos de tiempo de ejecución cuando se configura `DbEnable` en `true`.
- SageMaker Modelos compilados por AI NEO que este componente descarga automáticamente cuando se configura. `DeploymentEnable` `true`

Valor predeterminado: */greengrass/v2*`/work/aws.greengrass.SageMakerEdgeManager`

`DbEnable`

(Opcional) Puede habilitar este componente para almacenar los datos de tiempo de ejecución en una base de datos local para conservar los datos en caso de que el componente falle o el dispositivo se quede sin alimentación.

Esta base de datos requiere 5 MB de almacenamiento en el sistema de archivos del dispositivo principal.

Valor predeterminado: `false`

`DeploymentEnable`

Esta función está disponible en la versión 1.2.0 y en las versiones posteriores del componente SageMaker AI Edge Manager.

(Opcional) Puede habilitar este componente para recuperar automáticamente los modelos compilados por SageMaker AI NEO que cargue en Amazon S3. Después de cargar un modelo nuevo en Amazon S3, utilice SageMaker AI Studio o la API de SageMaker IA para implementar el nuevo modelo en este dispositivo principal. Al habilitar esta característica, puede implementar nuevos modelos en los dispositivos principales sin necesidad de crear una implementación de AWS IoT Greengrass .

> **Important**
>
> Para utilizar esta característica, debe configurar `DbEnable` en `true`. Esta característica utiliza la base de datos local para rastrear los modelos que recupera de la Nube de AWS.

Valor predeterminado: `false`

`DeploymentPollInterval`

Esta función está disponible en la versión 1.2.0 y en las versiones posteriores del componente SageMaker AI Edge Manager.

(Opcional) El tiempo (en minutos) entre el que este componente comprueba si hay nuevos modelos para descargar. Esta opción se aplica cuando se configura `DeploymentEnable` en `true`.

Valor predeterminado: `1440` (un día)

`DLRBackendOptions`

Esta función está disponible en la versión 1.2.0 y en las versiones posteriores del componente SageMaker AI Edge Manager.

(Opcional) El tiempo de ejecución del DLR marca el tiempo de ejecución del DLR que utiliza este componente. Puede utilizar la siguiente marca:

- `TVM_TENSORRT_CACHE_DIR`: habilita el almacenamiento en caché del modelo TensorRT. Especifique una ruta absoluta a una carpeta existente que tenga permisos de lectura/escritura.
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`: asigna el límite superior de la carpeta de caché del modelo TensorRT. Cuando el tamaño del directorio supera este límite, se eliminan los motores en caché que se utilizan menos. El valor predeterminado es 512 MB.

Por ejemplo, puede establecer este parámetro en el siguiente valor para habilitar el almacenamiento en caché del modelo TensorRT y limitar el tamaño de la caché a 800 MB.

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;
 TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

(Opcional) Valor de cadena que especifica si se debe habilitar el registro de depuración. Los valores admitidos son `true` y `false`.

Valor predeterminado: `false`

`UnixSocketName`

(Opcional) La ubicación del descriptor del archivo socket de SageMaker AI Edge Manager en el dispositivo principal.

Valor predeterminado: `/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica que el dispositivo principal forma parte del bucket de S3 *MyEdgeDeviceFleet* y que el agente escribe los datos de captura tanto en el dispositivo como en un bucket de S3. Esta configuración también permite el registro de depuración.

```
{
    "DeviceFleetName": "MyEdgeDeviceFleet",
    "BucketName": "amzn-s3-demo-bucket",
    "CaptureDataDestination": "Disk",
    "CaptureDataPeriodicUpload": "true",
    "SagemakerEdgeLogVerbose": "true"
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail
 10 -Wait
```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.3.6	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
1.3.5	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
1.3.4	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
1.3.3	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
1.3.2	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
1.3.1	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
1.3.0	Nuevas características • Suma compatibilidad con la administración del tamaño del disco caché de TensorRT. • Añade la `TVM_TENSORRT_CACHE_DISK_SIZE_MB` marca opcional al parámetro DLRBackend Options para establecer el límite de tamaño de los modelos almacenados en caché en el disco.

Versión	Cambios
	Mejoras • Proporciona una simultaneidad de predicciones mejorada. Esto ayuda a aprovechar mejor los motores de aceleración de dispositivos, como. GPUs
1.2.0	Nuevas características • Añade compatibilidad con este componente para recuperar automátic amente los modelos compilados por SageMaker AI NEO que usted carga en Amazon S3. Al habilitar esta función, puede implementar nuevos modelos en los dispositivos principales sin necesidad de crear una AWS IoT Greengrass implementación. • Suma compatibilidad con una base de datos de respaldo que este componente utiliza para conservar los datos de tiempo de ejecución, en caso de que el componente falle o el dispositivo se quede sin alimentac ión. • Suma compatibilidad para configurar los indicadores de tiempo de ejecución del DLR al configurar este componente.
1.1.1	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
1.1.0	Nuevas características • Suma compatibilidad con los dispositivos principales de Greengrass que ejecutan Amazon Linux 2. • Suma el nuevo parámetro de configuración `CaptureDataDiskPat h` . Puede usar este parámetro para especificar la ruta de la carpeta de datos capturados en su dispositivo. Mejoras y correcciones de errores • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
1.0.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.

Versión	Cambios
1.0.2	Mejoras y correcciones de errores Actualiza el script de instalación en el ciclo de vida del componente. Sus dispositivos principales ahora deben tener Python 3.6 o posterior, incluida `pip` para su versión de Python, instalado en el dispositivo antes de implementar este componente.
1.0.1	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
1.0.0	Versión inicial.

Clasificación de imágenes de DLR

El componente de clasificación de imágenes de DLR (`aws.greengrass.DLRImageClassification`) contiene un ejemplo de código de inferencia para realizar inferencias de clasificación de imágenes mediante el tiempo de ejecución de aprendizaje profundo y los modelos resnet-50. Este componente utiliza la variante Almacén de modelos de clasificación de imágenes de DLR y los componentes Tiempo de ejecución de DLR como dependencias para descargar el DLR y los modelos de muestra.

Para utilizar este componente de inferencia con un modelo de DLR entrenado de forma personalizada, cree una versión personalizada del componente almacén de modelos dependiente. Para usar su propio código de inferencia personalizado, puede usar la receta de este componente como plantilla para crear un componente de inferencia personalizado.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local

- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:

- NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

  ```
  pip3 install --upgrade numpy
  ```

- La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

  Cómo activar la pila de cámara antigua

  1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Seleccione Opciones de interfaz.
  3. Seleccione Cámara antigua para activar la pila de cámara antigua.
  4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.13 and 2.1.14

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.13 y 2.1.14 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.12

En la siguiente tabla, se muestran las dependencias de la versión 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.11

En la siguiente tabla, se muestran las dependencias de la versión 2.1.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.4 - 2.1.5

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.4 a 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	~2.0.0	Flexible
Almacén de modelos de clasificación de imágenes de DLR	~2.0.0	Rígido
DLR	~1.3.0	Flexible

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.1.x

`accessControl`

(Opcional) El objeto que contiene la política de autorización que permite al componente publicar mensajes en el tema de notificaciones predeterminado.

Predeterminado:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) El tema sobre el que desea publicar los resultados de la inferencia. Si modifica este valor, también debe modificar el valor del `resources` en el parámetro `accessControl` para que coincida con el nombre del tema personalizado.

Valor predeterminado: `ml/dlr/image-classification`

`Accelerator`

El acelerador que desea usar. Los valores admitidos son `cpu` y `gpu`.

Los modelos de ejemplo del componente del modelo dependiente solo admiten la aceleración de la CPU. Para usar la aceleración de la GPU con un modelo personalizado diferente, cree un componente de modelo personalizado para anular el componente del modelo público.

Valor predeterminado: `cpu`

`ImageDirectory`

(Opcional) La ruta de la carpeta del dispositivo donde los componentes de inferencia leen las imágenes. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso.

Valor predeterminado: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/image_classification/sample_images/`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`ImageName`

(Opcional) El nombre de la imagen que el componente de inferencia utiliza como entrada para realizar una predicción. El componente busca la imagen en la carpeta especificada en `ImageDirectory`. De forma predeterminada, el componente usa la imagen de muestra en el directorio de imágenes predeterminado. AWS IoT Greengrass admite los siguientes formatos de imagen: `jpegjpg`,`png`, `ynpy`.

Valor predeterminado: `cat.jpeg`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`InferenceInterval`

(Opcional) El tiempo en segundos entre cada predicción realizada por el código de inferencia. El código de inferencia de ejemplo se ejecuta indefinidamente y repite sus predicciones en el intervalo de tiempo especificado. Por ejemplo, puede cambiarlo por un intervalo más corto si desea utilizar imágenes tomadas por una cámara para realizar predicciones en tiempo real.

Valor predeterminado: `3600`

`ModelResourceKey`

(Opcional) Los modelos que se utilizan en el componente de modelo público dependiente. Modifique este parámetro solo si anuló el componente del modelo público por un componente personalizado.

Predeterminado:

```
{
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "windows": "DLR-resnet50-win-cpu-ImageClassification"
}
```

`UseCamera`

(Opcional) El valor de cadena que define si se deben utilizar imágenes de una cámara conectada al dispositivo principal de Greengrass. Los valores admitidos son `true` y `false`.

Si establece este valor en `true`, el código de inferencia de muestra accede a la cámara del dispositivo y ejecuta la inferencia localmente en la imagen capturada. Los valores de los parámetros `ImageName` y `ImageDirectory` se pasan por alto. Asegúrese de que el usuario que ejecuta este componente tenga read/write acceso a la ubicación en la que la cámara almacena las imágenes capturadas.

Valor predeterminado: `false`

> **Note**
>
> Al ver la receta de este componente, el parámetro `UseCamera` de configuración no aparece en la configuración predeterminada. Sin embargo, puede modificar el valor de este parámetro en una actualización de la combinación de configuraciones al implementar el componente.
> Si establece `UseCamera` en `true`, también debe crear un enlace simbólico para permitir que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución. Para obtener más información sobre cómo usar una cámara con los componentes de inferencia de muestra, consulte Actualización de las configuraciones de los componentes.

2.0.x

`MLRootPath`

(Opcional) La ruta de la carpeta en los dispositivos principales de Linux donde los componentes de inferencia leen las imágenes y escriben los resultados de la inferencia. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso el usuario que ejecuta este componente.

Valor predeterminado: *`/greengrass/v2`*`/work/variant.DLR/greengrass_ml`

Valor predeterminado: *`/greengrass/v2`*`/work/variant.TensorFlowLite/greengrass_ml`

`Accelerator`

El acelerador que desea usar. Los valores admitidos son `cpu` y `gpu`.

Los modelos de ejemplo del componente del modelo dependiente solo admiten la aceleración de la CPU. Para usar la aceleración de la GPU con un modelo personalizado diferente, cree un componente de modelo personalizado para anular el componente del modelo público.

Valor predeterminado: `cpu`

`ImageName`

(Opcional) El nombre de la imagen que el componente de inferencia utiliza como entrada para realizar una predicción. El componente busca la imagen en la carpeta especificada en `ImageDirectory`. La ubicación predeterminada es*`MLRootPath`*`/images`. AWS IoT Greengrass admite los siguientes formatos de imagen: `jpegjpg`,`png`, `ynpy`.

Valor predeterminado: `cat.jpeg`

`InferenceInterval`

(Opcional) El tiempo en segundos entre cada predicción realizada por el código de inferencia. El código de inferencia de ejemplo se ejecuta indefinidamente y repite sus predicciones en el intervalo de tiempo especificado. Por ejemplo, puede cambiarlo por un intervalo más corto si desea utilizar imágenes tomadas por una cámara para realizar predicciones en tiempo real.

Valor predeterminado: `3600`

`ModelResourceKey`

(Opcional) Los modelos que se utilizan en el componente de modelo público dependiente. Modifique este parámetro solo si anuló el componente del modelo público por un componente personalizado.

Predeterminado:

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log -
  Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.14	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.13	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.12	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.5	Componente publicado en su totalidad. Regiones de AWS
2.1.4	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass. Esta versión no está disponible en Europa (Londres) (`eu-west-2`).

Versión	Cambios
2.1.3	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.1	Nuevas características • Utilice la versión 1.6.0 del Tiempo de ejecución de aprendizaje profundo. • Agregue soporte para la clasificación de imágenes de muestra en las plataformas Armv8 (AArch64). Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano. • Habilite la integración de la cámara para la inferencia de muestras. Utilice el nuevo parámetro de configuración `UseCamera` para permitir que el código de inferencia de muestra acceda a la cámara del dispositivo principal de Greengrass y ejecute la inferencia localmente en la imagen capturada. • Suma compatibilidad para publicar los resultados de la inferencia en la Nube de AWS. Utilice el nuevo parámetro de configuración `PublishResultsOnTopic` para especificar el tema en el que desea publicar los resultados. • Agregue el nuevo parámetro de configuración `ImageDirectory` que le permite especificar un directorio personalizado para la imagen en la que desea realizar la inferencia. Mejoras y correcciones de errores • Escriba los resultados de la inferencia en el archivo de registro del componente en lugar de en un archivo de inferencia independiente. • Utilice el módulo de registro del software AWS IoT Greengrass Core para registrar la salida de los componentes. • Utilice el SDK para dispositivos con AWS IoT para leer la configuración del componente y aplicar los cambios de configuración.

Versión	Cambios
2.0.4	Versión inicial.

Detección de objetos del DLR

El componente de detección de objetos del DLR (`aws.greengrass.DLRObjectDetection`) contiene un ejemplo de código de inferencia para realizar inferencias de detección de objetos mediante el tiempo de ejecución de aprendizaje profundo y ejemplos de modelos previamente entrenados. Este componente utiliza la variante Almacén de modelos de detección de objetos del DLR y los componentes Tiempo de ejecución de DLR como dependencias para descargar el DLR y los modelos de muestra.

Para utilizar este componente de inferencia con un modelo de DLR entrenado de forma personalizada, cree una versión personalizada del componente almacén de modelos dependiente. Para usar su propio código de inferencia personalizado, puede usar la receta de este componente como plantilla para crear un componente de inferencia personalizado.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

Cómo activar la pila de cámara antigua

1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

   ```
   sudo raspi-config
   ```

2. Seleccione Opciones de interfaz.
3. Seleccione Cámara antigua para activar la pila de cámara antigua.
4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.13 and 2.1.14

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.13 y 2.1.14 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.12

En la siguiente tabla, se muestran las dependencias de la versión 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.11

En la siguiente tabla, se muestran las dependencias de la versión 2.1.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.4 - 2.1.5

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.4 a 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	~2.0.0	Flexible
Almacén de modelos de detección de objetos del DLR	~2.0.0	Rígido
DLR	~1.3.0	Flexible

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.1.x

`accessControl`

(Opcional) El objeto que contiene la política de autorización que permite al componente publicar mensajes en el tema de notificaciones predeterminado.

Predeterminado:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) El tema sobre el que desea publicar los resultados de la inferencia. Si modifica este valor, también debe modificar el valor del `resources` en el parámetro `accessControl` para que coincida con el nombre del tema personalizado.

Valor predeterminado: `ml/dlr/object-detection`

`Accelerator`

El acelerador que desea usar. Los valores admitidos son `cpu` y `gpu`.

Los modelos de ejemplo del componente del modelo dependiente solo admiten la aceleración de la CPU. Para usar la aceleración de la GPU con un modelo personalizado diferente, cree un componente de modelo personalizado para anular el componente del modelo público.

Valor predeterminado: `cpu`

`ImageDirectory`

(Opcional) La ruta de la carpeta del dispositivo donde los componentes de inferencia leen las imágenes. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso.

Valor predeterminado: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/object_detection/sample_images/`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`ImageName`

(Opcional) El nombre de la imagen que el componente de inferencia utiliza como entrada para realizar una predicción. El componente busca la imagen en la carpeta especificada en `ImageDirectory`. De forma predeterminada, el componente usa la imagen de muestra en el directorio de imágenes predeterminado. AWS IoT Greengrass admite los siguientes formatos de imagen: `jpegjpg`,`png`, `ynpy`.

Valor predeterminado: `objects.jpg`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`InferenceInterval`

(Opcional) El tiempo en segundos entre cada predicción realizada por el código de inferencia. El código de inferencia de ejemplo se ejecuta indefinidamente y repite sus predicciones en el intervalo de tiempo especificado. Por ejemplo, puede cambiarlo por un intervalo más corto si desea utilizar imágenes tomadas por una cámara para realizar predicciones en tiempo real.

Valor predeterminado: `3600`

`ModelResourceKey`

(Opcional) Los modelos que se utilizan en el componente de modelo público dependiente. Modifique este parámetro solo si anuló el componente del modelo público por un componente personalizado.

Predeterminado:

```
{
    "armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",
    "aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",
    "x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",
    "windows": "DLR-resnet50-win-cpu-ObjectDetection"
}
```

`UseCamera`

(Opcional) El valor de cadena que define si se deben utilizar imágenes de una cámara conectada al dispositivo principal de Greengrass. Los valores admitidos son `true` y `false`.

Si establece este valor en `true`, el código de inferencia de muestra accede a la cámara del dispositivo y ejecuta la inferencia localmente en la imagen capturada. Los valores de los parámetros `ImageName` y `ImageDirectory` se pasan por alto. Asegúrese de que el usuario que ejecuta este componente tenga read/write acceso a la ubicación en la que la cámara almacena las imágenes capturadas.

Valor predeterminado: `false`

> **Note**
>
> Al ver la receta de este componente, el parámetro `UseCamera` de configuración no aparece en la configuración predeterminada. Sin embargo, puede modificar el valor de este parámetro en una actualización de la combinación de configuraciones al implementar el componente.
> Si establece `UseCamera` en `true`, también debe crear un enlace simbólico para permitir que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución. Para obtener más información sobre cómo usar una cámara con los componentes de inferencia de muestra, consulte Actualización de las configuraciones de los componentes.

2.0.x

`MLRootPath`

(Opcional) La ruta de la carpeta en los dispositivos principales de Linux donde los componentes de inferencia leen las imágenes y escriben los resultados de la inferencia. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso el usuario que ejecuta este componente.

Valor predeterminado: *`/greengrass/v2`*`/work/variant.DLR/greengrass_ml`

Valor predeterminado: *`/greengrass/v2`*`/work/variant.TensorFlowLite/greengrass_ml`

`Accelerator`

No lo modifique. Actualmente, el único valor admitido para el acelerador es `cpu`, porque los modelos de los componentes del modelo dependiente se compilan únicamente para el acelerador de la CPU.

`ImageName`

(Opcional) El nombre de la imagen que el componente de inferencia utiliza como entrada para realizar una predicción. El componente busca la imagen en la carpeta especificada en `ImageDirectory`. La ubicación predeterminada es*`MLRootPath`*`/images`. AWS IoT Greengrass admite los siguientes formatos de imagen: `jpegjpg`,`png`, `ynpy`.

Valor predeterminado: `objects.jpg`

`InferenceInterval`

(Opcional) El tiempo en segundos entre cada predicción realizada por el código de inferencia. El código de inferencia de ejemplo se ejecuta indefinidamente y repite sus predicciones en el intervalo de tiempo especificado. Por ejemplo, puede cambiarlo por un intervalo más corto si desea utilizar imágenes tomadas por una cámara para realizar predicciones en tiempo real.

Valor predeterminado: `3600`

`ModelResourceKey`

(Opcional) Los modelos que se utilizan en el componente de modelo público dependiente. Modifique este parámetro solo si anuló el componente del modelo público por un componente personalizado.

Predeterminado:

```
{
    armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
    x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10
   -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.14	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.13	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.12	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.5	Componente publicado en su totalidad. Regiones de AWS
2.1.4	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass. Esta versión no está disponible en Europa (Londres) (`eu-west-2`).
2.1.3	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.

Versión	Cambios
2.1.2	Mejoras y correcciones de errores • Corrige un problema de escalado de la imagen que provocaba que los recuadros delimitadores no fueran precisos en los resultados de la inferencia de detección de objetos del DLR de muestra.
2.1.1	Nuevas características • Utilice la versión 1.6.0 del Tiempo de ejecución de aprendizaje profundo. • Añada compatibilidad con la detección de objetos de muestra en las plataformas Armv8 (AArch64). Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano. • Habilite la integración de la cámara para la inferencia de muestras. Utilice el nuevo parámetro de configuración `UseCamera` para permitir que el código de inferencia de muestra acceda a la cámara del dispositivo principal de Greengrass y ejecute la inferencia localmente en la imagen capturada. • Suma compatibilidad para publicar los resultados de la inferencia en la Nube de AWS. Utilice el nuevo parámetro de configuración `PublishResultsOnTopic` para especificar el tema en el que desea publicar los resultados. • Agregue el nuevo parámetro de configuración `ImageDirectory` que le permite especificar un directorio personalizado para la imagen en la que desea realizar la inferencia. Mejoras y correcciones de errores • Escriba los resultados de la inferencia en el archivo de registro del componente en lugar de en un archivo de inferencia independiente. • Utilice el módulo de registro del software AWS IoT Greengrass Core para registrar la salida de los componentes. • Utilice el SDK para dispositivos con AWS IoT para leer la configuración del componente y aplicar los cambios de configuración.
2.0.4	Versión inicial.

Almacén de modelos de clasificación de imágenes de DLR

El almacén de modelos de clasificación de imágenes del DLR es un componente del modelo de aprendizaje automático que contiene ResNet -50 modelos previamente entrenados como artefactos de Greengrass. Los modelos previamente entrenados que se utilizan en este componente se obtienen del GluonCV Model Zoo y se compilan con AI Neo Deep Learning Runtime. SageMaker

El componente de inferencia de clasificación de imágenes de DLR utiliza este componente como una dependencia para el origen del modelo. Para utilizar un modelo de DLR entrenado de forma personalizada, cree una versión personalizada de este componente del modelo e incluya el modelo personalizado como un artefacto componente. Puede usar la receta de este componente como plantilla para crear componentes de modelo personalizados.

> **Note**
>
> El nombre del componente del almacén de modelos de clasificación de imágenes de DLR varía en función de su versión. El nombre del componente para la versión 2.1.x y las versiones posteriores es `variant.DLR.ImageClassification.ModelStore`. El nombre del componente de la versión 2.0.x es `variant.ImageClassification.ModelStore`.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x (`variant.DLR.ImageClassification.ModelStore`)

- 2.0.x (`variant.ImageClassification.ModelStore`)

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

- La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

  Cómo activar la pila de cámara antigua

  1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Seleccione Opciones de interfaz.
  3. Seleccione Cámara antigua para activar la pila de cámara antigua.
  4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.12 - 2.1.14

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.12 y 2.1.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.1.11

En la siguiente tabla, se muestran las dependencias de la versión 2.1.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	~2.0.0	Flexible

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente no genera registros.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.13	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.12	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.

Versión	Cambios
2.1.11	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.5	Nuevas características • Suma ejemplos de modelos de clasificación de imágenes para los dispositivos principales de Windows. • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.

Versión	Cambios
2.1.1	Nuevas características • Añada un ejemplo de modelo de clasificación de imágenes de ResNet -50 para las plataformas Armv8 (). AArch64 Esto amplía la compatibi lidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano.
2.0.4	Versión inicial.

Almacén de modelos de detección de objetos del DLR

El almacén de modelos de detección de objetos del DLR es un componente del modelo de aprendizaje automático que contiene YOLOv3 modelos previamente entrenados como artefactos de Greengrass. Los modelos de muestra utilizados en este componente se obtienen del GluonCV Model Zoo y se compilan con SageMaker AI Neo Deep Learning Runtime.

El componente de inferencia de detección de objetos del DLR utiliza este componente como dependencia del origen del modelo. Para utilizar un modelo de DLR entrenado de forma personalizada, cree una versión personalizada de este componente del modelo e incluya el modelo personalizado como un artefacto componente. Puede usar la receta de este componente como plantilla para crear componentes de modelo personalizados.

> **Note**
>
> El nombre del almacén de modelo de detección de objetos del DLR varía en función de su versión. El nombre del componente para la versión 2.1.x y las versiones posteriores es `variant.DLR.ObjectDetection.ModelStore`. El nombre del componente de la versión 2.0.x es `variant.ObjectDetection.ModelStore`.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos

- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.13 and 2.1.14

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.13 y 2.1.14 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.1.12

En la siguiente tabla, se muestran las dependencias de la versión 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.1.11

En la siguiente tabla, se muestran las dependencias de la versión 2.1.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.1.5 and 2.1.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.5 y 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	~2.0.0	Flexible

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente no genera registros.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.14	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.13	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.12	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.6	Añade un modelo de CPU para solucionar un problema en los dispositivos Armv8 (). AArch64
2.1.5	Nuevas características • Suma ejemplos de modelos de detección de objetos para los dispositivos principales de Windows. Mejoras y correcciones de errores • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.

Versión	Cambios
2.1.4	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.1	Nuevas características • Agregue un modelo de detección de YOLOv3 objetos de muestra para las plataformas Armv8 (). AArch64 Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano.
2.0.4	Versión inicial.

Tiempo de ejecución de DLR

El componente de tiempo de ejecución de DLR (`variant.DLR`) contiene un script que instala el tiempo de ejecución de aprendizaje profundo (DLR) y sus dependencias en un entorno virtual del dispositivo. Los componentes Clasificación de imágenes de DLR y Detección de objetos del DLR utilizan este componente como dependencia para instalar el DLR. La versión del componente 1.6.x instala el DLR versión 1.6.0 y la versión del componente 1.3.x instala el DLR versión 1.3.0.

Para usar un tiempo de ejecución diferente, puede usar la receta de este componente como plantilla para crear un componente de machine learning personalizado.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración

- De uso
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 1.6.x
- 1.3.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.

Puntos de conexión y puertos

De forma predeterminada, este componente utiliza un script de instalación para instalar los paquetes mediante los comandos `apt`, `yum`, `brew` y `pip` en función de la plataforma que utilice el dispositivo principal. Este componente debe poder realizar solicitudes salientes a varios índices y repositorios de paquetes para ejecutar el script de instalación. Para permitir que el tráfico saliente de este componente pase por un proxy o firewall, debe identificar los puntos de conexión de los índices y repositorios de paquetes a los que se conecta el dispositivo principal para realizar la instalación.

Tenga en cuenta lo siguiente al identificar los puntos de conexión necesarios para el script de instalación de este componente:

- Los puntos de conexión dependen de la plataforma del dispositivo principal. Por ejemplo, un dispositivo principal que ejecuta Ubuntu usa `apt` en lugar de `yum` o `brew`. Además, los dispositivos que usan el mismo índice de paquetes pueden tener listas de orígenes diferentes, por lo que pueden recuperar paquetes de diferentes repositorios.

- Los puntos de conexión pueden diferir entre varios dispositivos que utilizan el mismo índice de paquetes, ya que cada dispositivo tiene sus propias listas de orígenes que definen dónde recuperar los paquetes.
- Los puntos de conexión pueden cambiar con el tiempo. Cada índice URLs de paquetes proporciona los repositorios en los que se descargan los paquetes, y el propietario de un paquete puede cambiar lo que proporciona URLs el índice de paquetes.

Para obtener más información sobre las dependencias que instala este componente y sobre cómo deshabilitar el script de instalación, consulte el UseInstallerparámetro de configuración.

Para obtener más información sobre los puntos de conexión y los puertos necesarios para el funcionamiento básico, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

1.6.11 - 1.6.16

En la siguiente tabla, se muestran las dependencias de las versiones 1.6.11 a 1.6.16 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible

1.6.10

En la siguiente tabla, se muestran las dependencias de la versión 1.6.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

1.6.9

En la siguiente tabla, se muestran las dependencias de la versión 1.6.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

1.6.8

En la siguiente tabla, se muestran las dependencias de la versión 1.6.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

1.6.6 and 1.6.7

En la siguiente tabla, se muestran las dependencias de las versiones 1.6.6 y 1.6.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

1.6.4 and 1.6.5

En la siguiente tabla, se muestran las dependencias de las versiones 1.6.4 y 1.6.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

1.6.3

En la siguiente tabla, se muestran las dependencias de la versión 1.6.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

1.6.2

En la siguiente tabla, se muestran las dependencias de la versión 1.6.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

1.6.1

En la siguiente tabla, se muestran las dependencias de la versión 1.6.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

1.3.x

En la siguiente tabla, se muestran las dependencias de la versión 1.3.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	~2.0.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`MLRootPath`

(Opcional) La ruta de la carpeta en los dispositivos principales de Linux donde los componentes de inferencia leen las imágenes y escriben los resultados de la inferencia. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso el usuario que ejecuta este componente.

Valor predeterminado: */greengrass/v2*`/work/variant.DLR/greengrass_ml`

`WindowsMLRootPath`

Esta característica está disponible en la versión 1.6.6 y posteriores de este componente.

(Opcional) La ruta de la carpeta del dispositivo principal de Windows donde los componentes de inferencia leen las imágenes y escriben los resultados de la inferencia. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso el usuario que ejecuta este componente.

Valor predeterminado: *C:\greengrass\v2*`\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(Opcional) Valor de cadena que define si se debe utilizar el script de instalación de este componente para instalar el DLR y sus dependencias. Los valores admitidos son `true` y `false`.

Establezca este valor en `false` si desea utilizar un script personalizado para la instalación del DLR o si desea incluir las dependencias del tiempo de ejecución en una imagen de Linux prediseñada. Para usar este componente con los componentes AWS de inferencia de DLR proporcionados, instale las siguientes bibliotecas, incluidas las dependencias, y póngalas `ggc_user` a disposición del usuario del sistema, por ejemplo, el que ejecuta los componentes de ML.

- Python 3.7 o posterior, incluido `pip` para su versión de Python.
- Versión 1.6.0 del tiempo de ejecución de aprendizaje profundo

- NumPy.
- OpenCV-Python.
- SDK para dispositivos con AWS IoT v2 para Python.
- AWS Python en tiempo de ejecución común (CRT).
- Picamera (solo para dispositivos Raspberry Pi).
- `awscam`módulo (para AWS DeepLens dispositivos).
- libGL (para dispositivos Linux)

Valor predeterminado: `true`

De uso

Utilice este componente con el parámetro de configuración `UseInstaller` establecido en `true` para instalar el DLR y sus dependencias en el dispositivo. El componente configura un entorno virtual en el dispositivo que incluye el OpenCV NumPy y las bibliotecas necesarias para el DLR.

> **Note**
>
> El script de instalación de este componente también instala las versiones más recientes de las bibliotecas de sistema adicionales necesarias para configurar el entorno virtual en el dispositivo y utilizar el marco de machine learning instalado. Esto podría actualizar las bibliotecas del sistema existentes en el dispositivo. Consulte la siguiente tabla para ver la lista de bibliotecas que instala este componente para cada sistema operativo compatible. Si desea personalizar este proceso de instalación, defina el parámetro de configuración `UseInstaller` en `false` y desarrolle su propio script de instalación.

Plataforma	Bibliotecas instaladas en el sistema del dispositivo	Bibliotecas instaladas en el entorno virtual
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	Ninguno
Ubuntu	`wget`	Ninguno

Al implementar el componente de inferencia, este componente de tiempo de ejecución comprueba primero si el dispositivo ya tiene instalado el DLR y sus dependencias y, de no ser así, los instala por usted.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la AWS IoT Greengrass carpeta raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.DLR.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.6.16	Versión actualizada para la versión 2.12.5 del núcleo de Greengrass.

Versión	Cambios
1.6.12	Mejoras y correcciones de errores • Corrige el script de instalación para los usuarios del sistema operativo Windows.
1.6.11	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
1.6.10	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
1.6.9	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
1.6.8	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
1.6.7	Mejoras y correcciones de errores • Actualiza el script de instalación `UseInstaller` para instalar libGL, que no está disponible de forma predeterminada en determinadas plataformas de Linux. • Actualiza el script de instalación `UseInstaller` para usar siempre Python 3.9 en el entorno virtual de este componente. Este cambio ayuda a garantizar la compatibilidad con otras bibliotecas.
1.6.6	Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Agrega el nuevo parámetro de configuración `WindowsMLRootPath` que puede usar para configurar la carpeta de resultados de inferencias en los dispositivos principales de Windows.
1.6.5	Nuevas características • Agrega el nuevo parámetro de configuración `UseInstaller` que puede usar para deshabilitar el script de instalación en este component e.

Versión	Cambios
1.6.4	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
1.6.3	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
1.6.2	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
1.6.1	Nuevas características • Instala la versión 1.6.0 del tiempo de ejecución de aprendizaje profundo y sus dependencias. • Se agregó soporte para instalar DLR en plataformas Armv8 (). AArch64 Esto amplía la compatibilidad con machine learning para los dispositivos principales de Greengrass que ejecutan NVIDIA Jetson, como el Jetson Nano. Mejoras y correcciones de errores • Instálelo SDK para dispositivos con AWS IoT en el entorno virtual para leer la configuración del componente y aplicar los cambios de configura ción. • Correcciones de errores y mejoras menores adicionales.
1.3.2	Versión inicial. Instala DLR versión 1.3.0.

TensorFlow Clasificación de imágenes Lite

El componente de clasificación de imágenes de TensorFlow Lite (`aws.greengrass.TensorFlowLiteImageClassification`) contiene un ejemplo de código de inferencia para realizar inferencias de clasificación de imágenes mediante el tiempo de ejecución de TensorFlow Lite y un ejemplo de modelo cuantificado MobileNet 1.0 previamente entrenado. Este componente utiliza la variante TensorFlow Tienda de modelos de clasificación de imágenes Lite y los TensorFlow Tiempo de ejecución Lite componentes como dependencias para descargar el motor de ejecución de TensorFlow Lite y el modelo de muestra.

Para usar este componente de inferencia con un modelo TensorFlow Lite personalizado, cree una versión personalizada del componente de la tienda de modelos dependiente. Para usar su propio código de inferencia personalizado, puede usar la receta de este componente como plantilla para crear un componente de inferencia personalizado.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se

enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.11 and 2.1.12

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.11 y 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`accessControl`

(Opcional) El objeto que contiene la política de autorización que permite al componente publicar mensajes en el tema de notificaciones predeterminado.

Predeterminado:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) El tema sobre el que desea publicar los resultados de la inferencia. Si modifica este valor, también debe modificar el valor del `resources` en el parámetro `accessControl` para que coincida con el nombre del tema personalizado.

Valor predeterminado: `ml/tflite/image-classification`

`Accelerator`

El acelerador que desea usar. Los valores admitidos son `cpu` y `gpu`.

Los modelos de ejemplo del componente del modelo dependiente solo admiten la aceleración de la CPU. Para usar la aceleración de la GPU con un modelo personalizado diferente, cree un componente de modelo personalizado para anular el componente del modelo público.

Valor predeterminado: `cpu`

`ImageDirectory`

(Opcional) La ruta de la carpeta del dispositivo donde los componentes de inferencia leen las imágenes. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso.

Valor predeterminado: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/image_classification/sample_images/`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`ImageName`

(Opcional) El nombre de la imagen que el componente de inferencia utiliza como entrada para realizar una predicción. El componente busca la imagen en la carpeta especificada en `ImageDirectory`. De forma predeterminada, el componente usa la imagen de muestra en el directorio de imágenes predeterminado. AWS IoT Greengrass admite los siguientes formatos de imagen: `jpegjpg`,`png`, `ynpy`.

Valor predeterminado: `cat.jpeg`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`InferenceInterval`

(Opcional) El tiempo en segundos entre cada predicción realizada por el código de inferencia. El código de inferencia de ejemplo se ejecuta indefinidamente y repite sus predicciones en el intervalo de tiempo especificado. Por ejemplo, puede cambiarlo por un intervalo más corto si desea utilizar imágenes tomadas por una cámara para realizar predicciones en tiempo real.

Valor predeterminado: `3600`

`ModelResourceKey`

(Opcional) Los modelos que se utilizan en el componente de modelo público dependiente. Modifique este parámetro solo si anuló el componente del modelo público por un componente personalizado.

Predeterminado:

```
{
    "model": "TensorFlowLite-Mobilenet"
}
```

`UseCamera`

(Opcional) El valor de cadena que define si se deben utilizar imágenes de una cámara conectada al dispositivo principal de Greengrass. Los valores admitidos son `true` y `false`.

Si establece este valor en `true`, el código de inferencia de muestra accede a la cámara del dispositivo y ejecuta la inferencia localmente en la imagen capturada. Los valores de los parámetros `ImageName` y `ImageDirectory` se pasan por alto. Asegúrese de que el usuario que ejecuta este componente tenga read/write acceso a la ubicación en la que la cámara almacena las imágenes capturadas.

Valor predeterminado: `false`

> **Note**
>
> Al ver la receta de este componente, el parámetro `UseCamera` de configuración no aparece en la configuración predeterminada. Sin embargo, puede modificar el valor de este parámetro en una actualización de la combinación de configuraciones al implementar el componente.
> Si establece `UseCamera` en `true`, también debe crear un enlace simbólico para permitir que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución. Para obtener más información sobre cómo usar una cámara con los componentes de inferencia de muestra, consulte Actualización de las configuraciones de los componentes.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.12	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.

Versión	Cambios
2.1.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Versión inicial.

TensorFlow Detección de objetos Lite

El componente de detección de objetos TensorFlow Lite (`aws.greengrass.TensorFlowLiteObjectDetection`) contiene un ejemplo de código de inferencia para realizar inferencias de detección de objetos con TensorFlow Lite y un ejemplo del modelo de detección de un solo disparo (SSD) MobileNet 1.0 previamente entrenado. Este componente utiliza la variante TensorFlow Tienda de modelos de detección de objetos Lite y los TensorFlow Tiempo de ejecución Lite componentes como dependencias para descargar TensorFlow Lite y el modelo de muestra.

Para usar este componente de inferencia con un modelo TensorFlow Lite personalizado, puede crear una versión personalizada del componente de la tienda de modelos dependiente. Para usar su propio código de inferencia personalizado, use la receta de este componente como plantilla para crear un componente de inferencia personalizado.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).

- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:

  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.11 and 2.1.12

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.11 y 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible
TensorFlow Tienda de modelos de clasificación de imágenes Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Lite	>=2.5.0 <2.6.0	Rígido

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`accessControl`

(Opcional) El objeto que contiene la política de autorización que permite al componente publicar mensajes en el tema de notificaciones predeterminado.

Predeterminado:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) El tema sobre el que desea publicar los resultados de la inferencia. Si modifica este valor, también debe modificar el valor del `resources` en el parámetro `accessControl` para que coincida con el nombre del tema personalizado.

Valor predeterminado: `ml/tflite/object-detection`

`Accelerator`

El acelerador que desea usar. Los valores admitidos son `cpu` y `gpu`.

Los modelos de ejemplo del componente del modelo dependiente solo admiten la aceleración de la CPU. Para usar la aceleración de la GPU con un modelo personalizado diferente, cree un componente de modelo personalizado para anular el componente del modelo público.

Valor predeterminado: `cpu`

`ImageDirectory`

(Opcional) La ruta de la carpeta del dispositivo donde los componentes de inferencia leen las imágenes. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso.

Valor predeterminado: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/object_detection/sample_images/`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`ImageName`

(Opcional) El nombre de la imagen que el componente de inferencia utiliza como entrada para realizar una predicción. El componente busca la imagen en la carpeta especificada en `ImageDirectory`. De forma predeterminada, el componente usa la imagen de muestra en el directorio de imágenes predeterminado. AWS IoT Greengrass admite los siguientes formatos de imagen: `jpegjpg`,`png`, `ynpy`.

Valor predeterminado: `objects.jpg`

> **Note**
>
> Si establece el valor de `UseCamera` en `true`, se ignora este parámetro de configuración.

`InferenceInterval`

(Opcional) El tiempo en segundos entre cada predicción realizada por el código de inferencia. El código de inferencia de ejemplo se ejecuta indefinidamente y repite sus predicciones en el intervalo de tiempo especificado. Por ejemplo, puede cambiarlo por un intervalo más corto si desea utilizar imágenes tomadas por una cámara para realizar predicciones en tiempo real.

Valor predeterminado: `3600`

`ModelResourceKey`

(Opcional) Los modelos que se utilizan en el componente de modelo público dependiente. Modifique este parámetro solo si anuló el componente del modelo público por un componente personalizado.

Predeterminado:

```
{
    "model": "TensorFlowLite-SSD"
```

```
}
```

`UseCamera`

(Opcional) El valor de cadena que define si se deben utilizar imágenes de una cámara conectada al dispositivo principal de Greengrass. Los valores admitidos son `true` y `false`.

Si establece este valor en `true`, el código de inferencia de muestra accede a la cámara del dispositivo y ejecuta la inferencia localmente en la imagen capturada. Los valores de los parámetros `ImageName` y `ImageDirectory` se pasan por alto. Asegúrese de que el usuario que ejecuta este componente tenga read/write acceso a la ubicación en la que la cámara almacena las imágenes capturadas.

Valor predeterminado: `false`

> **Note**
>
> Al ver la receta de este componente, el parámetro `UseCamera` de configuración no aparece en la configuración predeterminada. Sin embargo, puede modificar el valor de este parámetro en una actualización de la combinación de configuraciones al implementar el componente.
> Si establece `UseCamera` en `true`, también debe crear un enlace simbólico para permitir que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución. Para obtener más información sobre cómo usar una cámara con los componentes de inferencia de muestra, consulte Actualización de las configuraciones de los componentes.

> **Note**
>
> Al ver la receta de este componente, el parámetro `UseCamera` de configuración no aparece en la configuración predeterminada. Sin embargo, puede modificar el valor de este parámetro en una actualización de la combinación de configuraciones al implementar el componente.
> Si establece `UseCamera` en `true`, también debe crear un enlace simbólico para permitir que el componente de inferencia acceda a la cámara desde el entorno virtual creado por el componente de tiempo de ejecución. Para obtener más información sobre cómo usar una cámara con los componentes de inferencia de muestra, consulte Actualización de las configuraciones de los componentes.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.12	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.

Versión	Cambios
2.1.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Mejoras y correcciones de errores • Corrige un problema de escalado de la imagen que provocaba que los recuadros delimitadores no fueran precisos en los resultados de la inferencia de detección de objetos de TensorFlow Lite de muestra.
2.1.0	Versión inicial.

TensorFlow Tienda de modelos de clasificación de imágenes Lite

El almacén de modelos de clasificación de imágenes TensorFlow Lite (`variant.TensorFlowLite.ImageClassification.ModelStore`) es un componente del modelo de aprendizaje automático que contiene un modelo MobileNet v1 previamente entrenado como artefacto de Greengrass. El modelo de muestra utilizado en este componente se obtiene del TensorFlowHub y se implementa con Lite. TensorFlow

El componente TensorFlow Clasificación de imágenes Lite de inferencia usa este componente como una dependencia para el origen del modelo. Para usar un modelo TensorFlow Lite personalizado, cree una versión personalizada de este componente del modelo e incluya su modelo personalizado como un artefacto componente. Puede usar la receta de este componente como plantilla para crear componentes de modelo personalizados.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.

3. Seleccione Cámara antigua para activar la pila de cámara antigua.
4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.11 and 2.1.12

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.11 y 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente no genera registros.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.12	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.

Versión	Cambios
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Versión inicial.

TensorFlow Tienda de modelos de detección de objetos Lite

El almacén de modelos de detección de objetos TensorFlow Lite (`variant.TensorFlowLite.ObjectDetection.ModelStore`) es un componente del modelo de aprendizaje automático que contiene un modelo de detección de disparo único (SSD) previamente entrenado como un MobileNet artefacto de Greengrass. El modelo de muestra utilizado en este componente se obtiene del TensorFlow Hub y se implementa con Lite. TensorFlow

El componente de inferencia de detección de objetos de TensorFlow Lite utiliza este componente como una dependencia para la fuente del modelo. Para usar un modelo TensorFlow Lite personalizado, cree una versión personalizada de este componente del modelo e incluya su modelo personalizado como un artefacto componente. Puede usar la receta de este componente como plantilla para crear componentes de modelo personalizados.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

```
pip3 install --upgrade numpy
```

- La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

  Cómo activar la pila de cámara antigua

  1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Seleccione Opciones de interfaz.
  3. Seleccione Cámara antigua para activar la pila de cámara antigua.
  4. Reinicie el Raspberry Pi.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.11 and 2.1.12

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.11 y 2.1.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente no genera registros.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.12	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.1.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.10	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.

Versión	Cambios
2.1.7	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.1.0	Versión inicial.

TensorFlow Tiempo de ejecución Lite

El componente de tiempo de ejecución de TensorFlow Lite (`variant.TensorFlowLite`) contiene un script que instala la versión 2.5.0 de TensorFlow Lite y sus dependencias en un entorno virtual del dispositivo. El TensorFlow componente de clasificación de imágenes y detección de objetos de TensorFlow Lite utilizan este componente de tiempo de ejecución como una dependencia para instalar Lite. TensorFlow

> **Note**
>
> TensorFlow El componente de tiempo de ejecución de Lite, versión 2.5.6 y versiones posteriores, reinstala las instalaciones existentes del entorno de ejecución de TensorFlow

Lite y sus dependencias. Esta reinstalación ayuda a garantizar que el dispositivo principal ejecute versiones compatibles de Lite y sus dependencias. TensorFlow

Para usar un tiempo de ejecución diferente, puede usar la receta de este componente como plantilla para crear un componente de machine learning personalizado.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- De uso
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.5.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux

- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.

Puntos de conexión y puertos

De forma predeterminada, este componente utiliza un script de instalación para instalar los paquetes mediante los comandos `apt`, `yum`, `brew` y `pip` en función de la plataforma que utilice el dispositivo principal. Este componente debe poder realizar solicitudes salientes a varios índices y repositorios de paquetes para ejecutar el script de instalación. Para permitir que el tráfico saliente de este componente pase por un proxy o firewall, debe identificar los puntos de conexión de los índices y repositorios de paquetes a los que se conecta el dispositivo principal para realizar la instalación.

Tenga en cuenta lo siguiente al identificar los puntos de conexión necesarios para el script de instalación de este componente:

- Los puntos de conexión dependen de la plataforma del dispositivo principal. Por ejemplo, un dispositivo principal que ejecuta Ubuntu usa `apt` en lugar de `yum` o `brew`. Además, los dispositivos que usan el mismo índice de paquetes pueden tener listas de orígenes diferentes, por lo que pueden recuperar paquetes de diferentes repositorios.
- Los puntos de conexión pueden diferir entre varios dispositivos que utilizan el mismo índice de paquetes, ya que cada dispositivo tiene sus propias listas de orígenes que definen dónde recuperar los paquetes.
- Los puntos de conexión pueden cambiar con el tiempo. Cada índice URLs de paquetes proporciona los repositorios en los que se descargan los paquetes, y el propietario de un paquete puede cambiar lo que proporciona URLs el índice de paquetes.

Para obtener más información sobre las dependencias que instala este componente y sobre cómo deshabilitar el script de instalación, consulte el UseInstallerparámetro de configuración.

Para obtener más información sobre los puntos de conexión y los puertos necesarios para el funcionamiento básico, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia.

También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.5.14 and 2.5.15

En la siguiente tabla, se muestran las dependencias de las versiones 2.5.14 y 2.5.15 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

2.5.13

En la siguiente tabla, se muestran las dependencias de la versión 2.5.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

2.5.12

En la siguiente tabla, se muestran las dependencias de la versión 2.5.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

2.5.11

En la siguiente tabla, se muestran las dependencias de la versión 2.5.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

2.5.10

En la siguiente tabla, se muestran las dependencias de la versión 2.5.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

2.5.9

En la siguiente tabla, se muestran las dependencias de la versión 2.5.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

2.5.8

En la siguiente tabla, se muestran las dependencias de la versión 2.5.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

2.5.5 - 2.5.7

En la siguiente tabla, se muestran las dependencias de las versiones 2.5.5 a 2.5.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

2.5.3 and 2.5.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.5.3 y 2.5.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.5.2

En la siguiente tabla, se muestran las dependencias de la versión 2.5.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.5.1

En la siguiente tabla, se muestran las dependencias de la versión 2.5.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.5.0

En la siguiente tabla, se muestran las dependencias de la versión 2.5.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`MLRootPath`

(Opcional) La ruta de la carpeta en los dispositivos principales de Linux donde los componentes de inferencia leen las imágenes y escriben los resultados de la inferencia. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso el usuario que ejecuta este componente.

Valor predeterminado: *`/greengrass/v2`*`/work/variant.TensorFlowLite/greengrass_ml`

`WindowsMLRootPath`

Esta característica está disponible en la versión 1.6.6 y posteriores de este componente.

(Opcional) La ruta de la carpeta del dispositivo principal de Windows donde los componentes de inferencia leen las imágenes y escriben los resultados de la inferencia. Puede modificar este valor en cualquier ubicación del dispositivo a la que tenga read/write acceso el usuario que ejecuta este componente.

Valor predeterminado: *`C:\greengrass\v2`*`\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(Opcional) Valor de cadena que define si se debe utilizar el script de instalación de este componente para instalar TensorFlow Lite y sus dependencias. Los valores admitidos son `true` y `false`.

Establezca este valor `false` si desea utilizar un script personalizado para la instalación de TensorFlow Lite o si desea incluir las dependencias del tiempo de ejecución en una imagen de Linux prediseñada. Para usar este componente con los componentes AWS de inferencia de TensorFlow Lite proporcionados, instale las siguientes bibliotecas, incluidas las dependencias, y póngalas a disposición del usuario del sistema, por ejemplo, el que ejecuta los componentes de `ggc_user` ML.

- Python 3.8 o posterior, incluso `pip` para su versión de Python
- TensorFlow Lite v2.5.0
- NumPy
- OpenCV-Python
- SDK para dispositivos con AWS IoT v2 para Python
- AWS Python en tiempo de ejecución común (CRT)
- Picamera (para dispositivos Raspberry Pi)

- awscammódulo (para AWS DeepLens dispositivos)
- libGL (para dispositivos Linux)

Valor predeterminado: `true`

De uso

Utilice este componente con el parámetro de `UseInstaller` configuración establecido en `true` para instalar TensorFlow Lite y sus dependencias en el dispositivo. El componente configura un entorno virtual en el dispositivo que incluye el OpenCV NumPy y las bibliotecas necesarias para Lite. TensorFlow

> **Note**
>
> El script de instalación de este componente también instala las versiones más recientes de las bibliotecas de sistema adicionales necesarias para configurar el entorno virtual en el dispositivo y utilizar el marco de machine learning instalado. Esto podría actualizar las bibliotecas del sistema existentes en el dispositivo. Consulte la siguiente tabla para ver la lista de bibliotecas que instala este componente para cada sistema operativo compatible. Si desea personalizar este proceso de instalación, defina el parámetro de configuración `UseInstaller` en `false` y desarrolle su propio script de instalación.

Plataforma	Bibliotecas instaladas en el sistema del dispositivo	Bibliotecas instaladas en el entorno virtual
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	Ninguno
Ubuntu	`wget`	Ninguno

Al implementar el componente de inferencia, este componente de tiempo de ejecución comprueba primero si el dispositivo ya tiene instalado TensorFlow Lite y sus dependencias. De lo contrario, el componente de tiempo de ejecución los instala automáticamente.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la AWS IoT Greengrass carpeta raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.TensorFlowLite.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.5.15	Versión actualizada del lanzamiento del núcleo de Greengrass 2.12.5.
2.5.14	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.

Versión	Cambios
2.5.13	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.5.12	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.5.11	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.5.10	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.5.9	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.5.8	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.5.7	Mejoras y correcciones de errores • Actualiza el script de instalación `UseInstaller` para instalar libGL, que no está disponible de forma predeterminada en determinadas plataformas de Linux. • Actualiza el script de instalación `UseInstaller` para usar siempre Python 3.9 en el entorno virtual de este componente. Este cambio ayuda a garantizar la compatibilidad con otras bibliotecas.
2.5.6	Mejoras y correcciones de errores • Actualiza este componente para instalar el último parche de TensorFlow Lite 2.5.0 (`tflite-runtime-2.5.0.post1`), de modo que pueda usar este componente con Python 3.9. Si este componente no puede instalar ese parche, se instala en su lugar `tflite-runtime-2.5.0`. • Actualiza este componente para volver a instalar las instalaciones existentes de TensorFlow Lite y sus dependencias. Este cambio ayuda a garantizar que el dispositivo principal ejecute versiones compatibles de TensorFlow Lite y sus dependencias.

Versión	Cambios
2.5.5	Nuevas características • Suma compatibilidad con los dispositivos principales que ejecutan Windows. • Agrega el nuevo parámetro de configuración `WindowsMLRootPath` que puede usar para configurar la carpeta de resultados de inferencias en los dispositivos principales de Windows.
2.5.4	Nuevas características • Agrega el nuevo parámetro de configuración `UseInstaller` que permite deshabilitar el script de instalación en este componente.
2.5.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.5.2	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.5.1	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.5.0	Versión inicial.

Adaptador de protocolo Modbus-RTU

El componente adaptador de protocolo Modbus-RTU (`aws.greengrass.Modbus`) recopila información de los dispositivos Modbus RTU locales.

Para solicitar información a un dispositivo Modbus RTU local con este componente, publique un mensaje en el tema al que está suscrito este componente. En el mensaje, especifique la solicitud de Modbus RTU que se va a enviar a un dispositivo. A continuación, este componente publica una respuesta que contiene el resultado de la solicitud de Modbus RTU.

> **Note**
>
> Este componente proporciona una funcionalidad similar a la del conector adaptador del protocolo Modbus RTU de la V1. AWS IoT Greengrass Para obtener más información,

consulte Conector del adaptador de protocolo Modbus RTU en la Guía para desarrolladores de AWS IoT Greengrass V1.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada
- Datos de salida
- Solicitudes y respuestas de Modbus RTU
- Archivo de registro local
- Licencias
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de Lambda (`aws.greengrass.lambda`). El núcleo de Greengrass ejecuta la función de Lambda de este componente mediante el componente lanzador de Lambda.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- Una conexión física entre el dispositivo AWS IoT Greengrass principal y los dispositivos Modbus. El dispositivo principal debe ser conectado físicamente a la red de Modbus RTU a través de un puerto de serie (por ejemplo, un puerto USB).
- Para recibir los datos de salida de este componente, debe combinar la siguiente actualización de configuración para el componente del enrutador de suscripción antiguo (`aws.greengrass.LegacySubscriptionRouter`) cuando implemente este componente. Esta configuración especifica el tema en el que este componente publica las respuestas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "component:aws.greengrass.Modbus",
        "subject": "modbus/adapter/response",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
  ```

```
      "id": "aws-greengrass-modbus",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
modbus:version",
      "subject": "modbus/adapter/response",
      "target": "cloud"
    }
  }
}
```

- *region*Sustitúyalo por el Región de AWS que utilice.
- *version*Sustitúyala por la versión de la función Lambda que ejecuta este componente. Para encontrar la versión de la función de Lambda, debe ver la receta de la versión de este componente que desee implementar. Abra la página de detalles de este componente en la consola de AWS IoT Greengrass y busque el par clave-valor de la función de Lambda. Este par clave-valor contiene el nombre y la versión de la función de Lambda.

> **Important**
>
> Debe actualizar la versión de la función de Lambda en el enrutador de suscripción antiguo cada vez que implemente este componente. Esto garantiza que utilice la versión correcta de la función de Lambda para la versión del componente que implemente.

Para obtener más información, consulte Crear implementaciones.

- Se admite la ejecución del adaptador de protocolo Modbus-RTU en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.11

En la siguiente tabla, se muestran las dependencias de la versión 2.1.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.16.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 =2.0.0 <2.15.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.14.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.4 and 2.1.5

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.4 y 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.8 and 2.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.8 y 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Rígido
Lanzador de Lambda	>=1.0.0	Rígido
Tiempos de ejecución de Lambda	>=1.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=1.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

> **Note**
>
> La configuración predeterminada de este componente incluye los parámetros de la función de Lambda. Le recomendamos que edite solo los siguientes parámetros para configurar este componente en sus dispositivos.

v2.1.x

`lambdaParams`

Un objeto que contiene los parámetros de la función de Lambda de este componente. Este objeto contiene la siguiente información:

`EnvironmentVariables`

Un objeto que contiene los parámetros de la función de Lambda. Este objeto contiene la siguiente información:

`ModbusLocalPort`

La ruta absoluta del puerto de serie físico de Modbus en el dispositivo principal, como `/dev/ttyS2`.

Para ejecutar este componente en un contenedor, debe definir esta ruta como un dispositivo del sistema (en `containerParams.devices`) al que pueda acceder el componente. Este componente se ejecuta en un contenedor de forma predeterminada.

> **Note**
>
> Este componente debe tener acceso al dispositivo. read/write

`ModbusBaudRate`

(Opcional) Un valor de cadena que especifica la velocidad en baudios para la comunicación en serie con los dispositivos Modbus TCP locales.

Valor predeterminado: `9600`

`ModbusByteSize`

(Opcional) Un valor de cadena que especifica el tamaño de un byte en la comunicación en serie con los dispositivos Modbus TCP locales. Elija `5`, `6`, `7` o `8` bits.

Valor predeterminado: `8`

`ModbusParity`

(Opcional) El modo de paridad que se utilizará para verificar la integridad de los datos en la comunicación en serie con los dispositivos Modbus TCP locales.

- `E`: verifique la integridad de los datos con una paridad uniforme.
- `0`: verifique la integridad de los datos con una paridad impar.
- `N`: no verifique la integridad de los datos.

Valor predeterminado: `N`

`ModbusStopBits`

(Opcional) Un valor de cadena que especifica el número de bits que indican el final de un byte en la comunicación en serie con los dispositivos Modbus TCP locales.

Valor predeterminado: `1`

`containerMode`

(Opcional) El modo de almacenamiento en contenedores de este componente. Puede elegir entre las siguientes opciones:

- `GreengrassContainer`— El componente se ejecuta en un entorno de ejecución aislado dentro del AWS IoT Greengrass contenedor.

Si especifica esta opción, debe especificar un dispositivo del sistema (en `containerParams.devices`) para que el contenedor acceda al dispositivo Modbus.

- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.

Valor predeterminado: `GreengrassContainer`

`containerParams`

(Opcional) Un objeto que contiene los parámetros de contenedor de este componente. El componente utiliza estos parámetros si se especifica `GreengrassContainer` para `containerMode`.

Este objeto contiene la siguiente información:

`memorySize`

(Opcional) La cantidad de memoria (en kilobytes) que se va a asignar al componente.

Predeterminado de 512 MB (525 312 KB).

`devices`

(Opcional) Objeto que especifica los dispositivos del sistema a los que puede acceder el componente en un contenedor.

> **Important**
>
> Para ejecutar este componente en un contenedor, debe especificar el dispositivo del sistema que configura en la variable de entorno `ModbusLocalPort`.

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`path`

La ruta al dispositivo del sistema en el dispositivo principal. Debe tener el mismo valor que el valor para el que se configure para `ModbusLocalPort`.

`permission`

(Opcional) El permiso para acceder al dispositivo del sistema desde el contenedor. Este valor debe ser `rw` el que especifique que el componente tiene read/write acceso al dispositivo del sistema.

Valor predeterminado: `rw`

`addGroupOwner`

(Opcional) Si desea agregar o no el grupo de sistemas que ejecuta el componente como propietario del dispositivo del sistema.

Valor predeterminado: `true`

`pubsubTopics`

(Opcional) Un objeto que contiene los temas a los que el componente se suscribe para recibir mensajes. Puede especificar cada tema y si el componente se suscribe a los temas de MQTT AWS IoT Core o a los temas locales publish/subscribe .

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`type`

(Opcional) El tipo de publish/subscribe mensajes que utiliza este componente para suscribirse a los mensajes. Puede elegir entre las siguientes opciones:

- `PUB_SUB` — Suscribirse a la mensajería de publicación/suscripción local. Si elige esta opción, el tema no podrá contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde un componente personalizado cuando especifique esta opción, consulte Publicar/suscribir mensajes locales.
- `IOT_CORE`— Suscríbase a los mensajes de AWS IoT Core MQTT. Si elige esta opción, el tema puede contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde componentes personalizados cuando especifique esta opción, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Valor predeterminado: `PUB_SUB`

`topic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `IotCore` para `type`, puede usar los comodines de MQTT (+ y #) en este tema.

Example Ejemplo: actualización de la combinación de configuraciones (modo en contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example Ejemplo: actualización de la combinación de configuraciones (modo sin contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

`lambdaParams`

Un objeto que contiene los parámetros de la función de Lambda de este componente. Este objeto contiene la siguiente información:

`EnvironmentVariables`

Un objeto que contiene los parámetros de la función de Lambda. Este objeto contiene la siguiente información:

`ModbusLocalPort`

La ruta absoluta del puerto de serie físico de Modbus en el dispositivo principal, como `/dev/ttyS2`.

Para ejecutar este componente en un contenedor, debe definir esta ruta como un dispositivo del sistema (en `containerParams.devices`) al que pueda acceder el componente. Este componente se ejecuta en un contenedor de forma predeterminada.

> **Note**
>
> Este componente debe tener read/write acceso al dispositivo.

`containerMode`

(Opcional) El modo de almacenamiento en contenedores de este componente. Puede elegir entre las siguientes opciones:

- `GreengrassContainer`— El componente se ejecuta en un entorno de ejecución aislado dentro del AWS IoT Greengrass contenedor.

  Si especifica esta opción, debe especificar un dispositivo del sistema (en `containerParams.devices`) para que el contenedor acceda al dispositivo Modbus.
- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.

Valor predeterminado: `GreengrassContainer`

`containerParams`

(Opcional) Un objeto que contiene los parámetros de contenedor de este componente. El componente utiliza estos parámetros si se especifica `GreengrassContainer` para `containerMode`.

Este objeto contiene la siguiente información:

`memorySize`

(Opcional) La cantidad de memoria (en kilobytes) que se va a asignar al componente.

Predeterminado de 512 MB (525 312 KB).

`devices`

(Opcional) Objeto que especifica los dispositivos del sistema a los que puede acceder el componente en un contenedor.

> **Important**
>
> Para ejecutar este componente en un contenedor, debe especificar el dispositivo del sistema que configura en la variable de entorno `ModbusLocalPort`.

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`path`

La ruta al dispositivo del sistema en el dispositivo principal. Debe tener el mismo valor que el valor para el que se configure para `ModbusLocalPort`.

`permission`

(Opcional) El permiso para acceder al dispositivo del sistema desde el contenedor. Este valor debe ser `rw` el que especifique que el componente tiene read/write acceso al dispositivo del sistema.

Valor predeterminado: `rw`

`addGroupOwner`

(Opcional) Si desea agregar o no el grupo de sistemas que ejecuta el componente como propietario del dispositivo del sistema.

Valor predeterminado: `true`

`pubsubTopics`

(Opcional) Un objeto que contiene los temas a los que el componente se suscribe para recibir mensajes. Puede especificar cada tema y si el componente se suscribe a los temas de MQTT AWS IoT Core o a los temas locales publish/subscribe .

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`type`

(Opcional) El tipo de publish/subscribe mensajes que utiliza este componente para suscribirse a los mensajes. Puede elegir entre las siguientes opciones:

- `PUB_SUB` — Suscribirse a la mensajería de publicación/suscripción local. Si elige esta opción, el tema no podrá contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde un componente personalizado cuando especifique esta opción, consulte Publicar/suscribir mensajes locales.
- `IOT_CORE`— Suscríbase a los mensajes de AWS IoT Core MQTT. Si elige esta opción, el tema puede contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde componentes personalizados cuando especifique esta opción, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Valor predeterminado: `PUB_SUB`

`topic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `IotCore` para `type`, puede usar los comodines de MQTT (+ y #) en este tema.

Example Ejemplo: actualización de la combinación de configuraciones (modo en contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example Ejemplo: actualización de la combinación de configuraciones (modo sin contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

Datos de entrada

Este componente acepta los parámetros de solicitud de Modbus RTU relacionados con el tema siguiente y envía la solicitud de Modbus RTU al dispositivo. De forma predeterminada, este componente se suscribe a los mensajes locales publish/subscribe . Para obtener más información sobre cómo publicar mensajes en este componente desde sus componentes personalizados, consulte Publicar/suscribir mensajes locales.

Tema predeterminado (publicación/suscripción local): `modbus/adapter/request`

El mensaje acepta las siguientes propiedades. Los mensajes de entrada deben tener un formato JSON válido.

`request`

Los parámetros de la solicitud de Modbus RTU que se van a enviar.

La forma del mensaje de solicitud depende del tipo de solicitud de Modbus RTU que representa. Las siguientes propiedades son necesarias para todas las solicitudes.

Tipo: `object` que contiene la siguiente información:

`operation`

El nombre de la operación que se va a ejecutar. Por ejemplo, especifique `ReadCoilsRequest` para leer las bobinas de un dispositivo Modbus RTU. Para obtener más información acerca de las operaciones permitidas, consulte Solicitudes y respuestas de Modbus RTU.

Tipo: `string`

`device`

El dispositivo de destino de la solicitud.

El valor debe ser un número entero entre `0` y `247`.

Tipo: `integer`

El resto de los parámetros que se incluirán en la solicitud dependen de la operación. Este componente gestiona la comprobación de redundancia cíclica (CRC) para verificar las solicitudes de datos por usted.

> **Note**
>
> Si la solicitud incluye una propiedad `address`, debe especificar su valor como un número entero. Por ejemplo, `"address": 1`.

`id`

Un ID arbitrario para la solicitud. Use esta propiedad para asignar una solicitud de entrada a una respuesta de salida. Si especifica esta propiedad, el componente establece la propiedad `id` en el objeto de respuesta para este valor.

Tipo: `string`

Example Ejemplo de entrada: Solicitud de lectura de salidas digitales (coils)

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

Datos de salida

Este componente publica las respuestas como datos de salida sobre el siguiente tema MQTT de forma predeterminada. Debe especificar este tema como parte de `subject` en la configuración del componente antiguo del enrutador de suscripciones. Para obtener más información sobre cómo suscribirse a los mensajes sobre este tema en sus componentes personalizados, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Tema predeterminado (AWS IoT Core MQTT): `modbus/adapter/response`

La forma del mensaje de respuesta depende de la operación de solicitud y del estado de la respuesta. Para ver ejemplos, consulte Solicitudes y respuestas de ejemplo.

Cada respuesta incluye las siguientes propiedades:

`response`

La respuesta del dispositivo Modbus RTU.

Tipo: `object` que contiene la siguiente información:

`status`

El estado de la solicitud. El estado puede ser uno de los siguientes valores:

- `Success`: la solicitud es válida, el componente envía la solicitud a la red de Modbus RTU y la red de Modbus RTU devuelve una respuesta.

- `Exception`: la solicitud es válida, el componente envía la solicitud a la red de Modbus RTU y la red de Modbus RTU devuelve una excepción. Para obtener más información, consulte Estado de respuesta: excepción.
- `No Response`: la solicitud no era válida y el componente detecta el error antes de que la solicitud se envíe a través de la red de Modbus RTU. Para obtener más información, consulte Estado de respuesta: sin respuesta.

`operation`

La operación que solicitó el componente.

`device`

El dispositivo al que el componente envió la solicitud.

`payload`

La respuesta del dispositivo Modbus RTU. Si `status` es `No Response`, este objeto contiene únicamente una propiedad `error` con la descripción de error (por ejemplo, `[Input/Output] No Response received from the remote unit`).

`id`

El identificador de la solicitud, que puede utilizar para identificar qué respuesta corresponde a qué solicitud.

> **Note**
>
> Una respuesta para una operación de escritura es simplemente un eco de la solicitud. Aunque no se devuelve información significante para las respuestas de escritura, se recomienda comprobar el estado de la respuesta para verificar si la solicitud tiene éxito o falla.

Example Ejemplo de salida: Correcto

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
```

```
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example Ejemplo de salida: Error

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "MyRequest"
}
```

Para obtener más ejemplos, consulte Solicitudes y respuestas de ejemplo.

Solicitudes y respuestas de Modbus RTU

Este conector acepta los parámetros de solicitud de Modbus RTU como datos de entrada y publica las respuestas como datos de salida.

Se admiten las siguientes operaciones comunes.

Nombre de la operación en la solicitud	Código de característica en la respuesta
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04

Nombre de la operación en la solicitud	Código de característica en la respuesta
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

Solicitudes y respuestas de ejemplo

A continuación, se muestran ejemplos de solicitudes y respuestas de las operaciones compatibles.

Leer bobinas

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
```

```
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

Leer entradas discretas

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "ReadDiscreteInputsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadDiscreteInputsRequest",
    "payload": {
      "function_code": 2,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

Leer registros mantenidos

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
```

```
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadHoldingRegistersRequest",
    "payload": {
      "function_code": 3,
      "registers": [20,30]
    }
  },
  "id" : "TestRequest"
}
```

Leer registros de entrada

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Escribir una única bobina

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "WriteSingleCoilRequest",
    "device": 1,
```

```
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteSingleCoilRequest",
    "payload": {
      "function_code": 5,
      "address": 1,
      "value": true
    }
  },
  "id" : "TestRequest"
}
```

Escribir un único registro

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Escribir varias bobinas

Ejemplo de solicitud:

```
{
  "request": {
```

```
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleCoilsRequest",
    "payload": {
      "function_code": 15,
      "address": 1,
      "count": 4
    }
  },
  "id" : "TestRequest"
}
```

Escribir varios registros

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "WriteMultipleRegistersRequest",
    "device": 1,
    "address": 1,
    "values": [20,30,10]
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
```

```
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "address": 1,
      "count": 3
    }
  },
  "id" : "TestRequest"
}
```

Enmascarar el registro de escritura

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "MaskWriteRegisterRequest",
    "device": 1,
    "address": 1,
    "and_mask": 175,
    "or_mask": 1
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    }
  },
  "id" : "TestRequest"
}
```

Leer y escribir varios registros

Ejemplo de solicitud:

```
{
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
    "read_count": 2,
    "write_address": 3,
    "write_registers": [20,30,40]
  },
  "id": "TestRequest"
}
```

Ejemplo de respuesta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadWriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "registers": [10,20,10,20]
    }
  },
  "id" : "TestRequest"
}
```

> **Note**
>
> La respuesta incluye los registros que lee el componente.

Estado de respuesta: excepción

Las excepciones pueden producirse cuando el formato de la solicitud es válido, pero la solicitud no se completó correctamente. En este caso, la respuesta contiene la siguiente información:

- `status` se establece en `Exception`.

- `function_code` equivale al código de la característica de la solicitud + 128.
- `exception_code` contiene el código de excepción. Para obtener más información, consulte los códigos de excepción de Modbus.

Ejemplo:

```
{
  "response": {
    "status": "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id": "TestRequest"
}
```

Estado de respuesta: sin respuesta

Este conector realiza las comprobaciones de validación de la solicitud de Modbus. Por ejemplo, comprueba los formatos no válidos y los campos que faltan. Si no se supera la validación, el conector no envía la solicitud. En su lugar, devuelve una respuesta que contiene la siguiente información:

- `status` se establece en `No Response`.
- `error` contiene el motivo del error.
- `error_message` contiene el mensaje de error.

Ejemplos:

```
{
  "response": {
    "status": "fail",
    "error_message": "Invalid address field. Expected <type 'int'>, got <type 'str'>",
    "error": "No Response",
    "device": 1,
```

```
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "Invalid address field. Expected Expected <type 'int'>, got <type
 'str'>"
    }
  },
  "id": "TestRequest"
}
```

Si la solicitud selecciona como destino un dispositivo inexistente o si la red de Modbus RTU no funciona, es posible que aparezca una respuesta `ModbusIOException`, que utiliza el formato de respuesta No.

```
{
  "response": {
    "status": "fail",
    "error_message": "[Input/Output] No Response received from the remote unit",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "[Input/Output] No Response received from the remote unit"
    }
  },
  "id": "TestRequest"
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyalo por la ruta a la carpeta AWS IoT Greengrass raíz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
  ```

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- Licencia pymodbus/BSD
- Licencia pyserial/BSD

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.11	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.1.10	Versión actualizada para la versión 2.14.0 de Greengrass nucleus.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.5	Mejoras y correcciones de errores • Corrige un problema con la operación `ReadDiscreteInput` .
2.1.4	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.

Versión	Cambios
2.1.2	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Agrega las opciones `ModbusBaudRate` , `ModbusByteSize` , `ModbusParity` y `ModbusStopBits` que se pueden especificar para configurar la comunicación en serie con los dispositivos Modbus RTU.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Puente MQTT

El componente puente MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`) transmite mensajes MQTT entre los dispositivos cliente, la publicación/suscripción local de Greengrass y. AWS IoT Core Puede utilizar este componente para actuar sobre los mensajes MQTT de los dispositivos de cliente en componentes personalizados y sincronizar los dispositivos de cliente con la Nube de AWS.

> **Note**
>
> Los dispositivos de cliente son dispositivos IoT locales que se conectan a un dispositivo principal de Greengrass para enviar mensajes MQTT y datos para su procesamiento. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Puede usar este componente para retransmitir mensajes entre los siguientes agentes de mensajes:

- MQTT local: el agente MQTT local maneja los mensajes entre los dispositivos de cliente y un dispositivo principal.
- Local publish/subscribe : el agente de mensajes local de Greengrass gestiona los mensajes entre los componentes de un dispositivo central. Para obtener más información acerca de cómo interactuar con estos mensajes en los componentes de Greengrass, consulte Publicar/suscribir mensajes locales.
- AWS IoT Core — El broker AWS IoT Core MQTT gestiona los mensajes entre dispositivos y Nube de AWS destinos de IoT. Para obtener más información acerca de cómo interactuar con estos mensajes en los componentes de Greengrass, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

  > **Note**
  >
  > El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias

- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- Si configura el componente agente MQTT del dispositivo principal para que utilice un puerto que no sea el puerto 8883 predeterminado, debe usar el puente de MQTT versión 2.1.0 o posterior. Configúrelo para que se conecte al puerto en el que opera el agente.
- Se admite la ejecución del componente puente de MQTT en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.3.2

En la siguiente tabla, se muestran las dependencias de la versión 2.3.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.6.0	Rígido

2.3.0 and 2.3.1

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.0 y 2.3.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.5.0	Rígido

2.2.5 and 2.2.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.5 y 2.2.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.5.0	Rígido

2.2.3 and 2.2.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.3 y 2.2.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.4.0	Rígido

2.2.0 – 2.2.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.0 a 2.2.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.3.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.0.0 <2.2.0	Rígido

2.0.0 to 2.1.0

En la siguiente tabla, se muestra las dependencias de las versiones 2.0.0 a 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.0.0 <2.1.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.3.0 – 2.3.2

`mqttTopicMapping`

Las asignaciones de temas que desea unir. Este componente se suscribe a los mensajes del tema de origen y publica los mensajes que recibe en el tema de destino. Cada asignación de temas define el tema, el tipo de origen y el tipo de destino.

Este objeto contiene la siguiente información:

`topicMappingNameKey`

El nombre de esta asignación de temas. *`topicMappingNameKey`*Sustitúyalo por un nombre que ayude a identificar el mapeo de este tema.

Este objeto contiene la siguiente información:

`topic`

El tema o filtro de temas que sirve de puente entre los agentes de origen y de destino.

Puede utilizar los caracteres comodín de temas MQTT + y # para transmitir mensajes en todos los temas que coincidan con un filtro de temas. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

> **Note**
>
> Para usar comodines de temas de MQTT con el agente `Pubsub` de origen, debe usar la versión 2.6.0 o posterior del componente núcleo de Greengrass.

`targetTopicPrefix`

El prefijo que se agrega al tema de destino cuando este componente transmite el mensaje.

`source`

El agente de mensajes de origen. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`target`

El agente de mensajes de destino. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

mqtt5 RouteOptions

(Opcional) Ofrece opciones para configurar las asignaciones de temas a fin de unir los mensajes del tema de origen al tema de destino.

Este objeto contiene la siguiente información:

`mqtt5RouteOptionsNameKey`

El nombre de las opciones de ruta de una asignación de temas. *`mqtt5RouteOptionsNameKey`*Sustitúyala por la coincidencia *`topicMappingNameKey`* definida en el `mqttTopicMapping` campo.

Este objeto contiene la siguiente información:

noLocal

(Opcional) Cuando está activado, el puente no reenvía los mensajes sobre un tema que haya publicado el propio puente. Utilice esta opción para evitar los bucles, de la siguiente manera:

```
{
```

```
    "mqtt5RouteOptions": {
        "toIoTCore": {
            "noLocal": true
        }
    },
    "mqttTopicMapping": {
        "toIoTCore": {
            "topic": "device",
            "source": "LocalMqtt",
            "target": "IotCore"
        },
        "toLocal": {
            "topic": "device",
            "source": "IotCore",
            "target": "LocalMqtt"
        }
    }
}
```

`noLocal` solo se admite en las rutas en las que `source` es `LocalMqtt`.

Predeterminado: false

retainAsPublished

(Opcional) Cuando está habilitado, los mensajes reenviados por el puente tienen la misma marca `retain` que los mensajes publicados en el agente para esa ruta.

`retainAsPublished` solo se admite en las rutas en las que `source` es `LocalMqtt`.

Predeterminado: false

mqtt

(Opcional) Configuración del protocolo MQTT para comunicarse con el agente local.

versión

(Opcional) La versión del protocolo MQTT utilizada por el puente para comunicarse con el agente local. Debe ser la misma que la versión de MQTT seleccionada en la configuración del núcleo.

Elija una de las siguientes opciones:

- `mqtt3`

- `mqtt5`

  Debe implementar un agente MQTT cuando el campo `source` o `target` del objeto `mqttTopicMapping` esté establecido en `LocalMqtt`. Si elige la opción `mqtt5`, debe usar el Agente MQTT 5 (EMQX).

  Valor predeterminado: `mqtt3`

ackTimeoutSeconds

(Opcional) Intervalo de tiempo para esperar a que lleguen los paquetes PUBACK, SUBACK o UNSUBACK antes de que se produzca un error en la operación.

Predeterminado: 60

connAckTimeoutSrta.

(Opcional) Intervalo de tiempo para esperar un paquete CONNACK antes de cerrar la conexión.

Valor predeterminado: 20 000 (20 segundos)

pingTimeoutMs

(Opcional) El tiempo en milisegundos que el puente espera a recibir un mensaje PINGACK del agente local. Si la espera supera el tiempo de espera, el puente se cierra y vuelve a abrir la conexión MQTT. Este valor debe ser inferior a `keepAliveTimeoutSeconds`.

Valor predeterminado: 30 000 (30 segundos)

keepAliveTimeoutSegundos

(Opcional) El tiempo en segundos que transcurre entre cada mensaje PING que envía el puente para mantener activa la conexión MQTT. El valor debe ser mayor que `pingTimeoutMs`.

Predeterminado: 60

maxReconnectDelaySra.

(Opcional) Tiempo máximo en segundos que MQTT se vuelve a conectar.

Valor predeterminado: 30 000 (30 segundos)

minReconnectDelaySra.

(Opcional) Tiempo mínimo en segundos para que MQTT se vuelva a conectar.

receiveMaximum

(Opcional) La cantidad máxima de paquetes QoS1 no confirmados que puede enviar el puente.

Predeterminado: 100

maximumPacketSize

El número máximo de bytes que el cliente aceptará para un paquete MQTT.

Predeterminado: nulo (sin límite)

sessionExpiryInterval

(Opcional) La cantidad de tiempo en segundos que puede solicitar para que dure una sesión entre el puente y el agente local.

Predeterminado: 4294967295 (la sesión nunca caduca)

`brokerUri`

(Opcional) El URI del agente MQTT local. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883. Utilice el siguiente formato y *`port`* sustitúyalo por el puerto en el que opera el broker MQTT:`ssl://localhost:`*`port`*.

Valor predeterminado: `ssl://localhost:8883`

startupTimeoutSeconds

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `BROKEN` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de archivo de configuración especifica lo siguiente:

- Transmita mensajes desde los dispositivos cliente a AWS IoT Core temas que coincidan con el filtro de `clients/+/hello/world` temas.
- Transmita los mensajes de los dispositivos cliente a los dispositivos locales publish/subscribe sobre los temas que coincidan con el `clients/+/detections` filtro de temas y añada el

`events/input/` prefijo al tema de destino. El tema de destino resultante coincide con el filtro de temas `events/input/clients/+/detections`.

- Transmita los mensajes desde los dispositivos cliente a AWS IoT Core temas que coincidan con el filtro de temas y añada el `$aws/rules/StatusUpdateRule/` prefijo al `clients/+/status` tema de destino. En este ejemplo, se transmiten estos mensajes directamente a una regla AWS IoT denominada `StatusUpdateRule` para la reducción de costos mediante la ingesta básica.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example Ejemplo: configuración de MQTT 5

El siguiente ejemplo de archivo de configuración actualiza lo siguiente:

- Permite que el puente utilice el protocolo MQTT 5 con el agente local.
- Configura MQTT para retenerlo como configuración publicada para la asignación de temas `ClientDeviceHelloWorld`.

```
{
```

```
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

`mqttTopicMapping`

Las asignaciones de temas que desea unir. Este componente se suscribe a los mensajes del tema de origen y publica los mensajes que recibe en el tema de destino. Cada asignación de temas define el tema, el tipo de origen y el tipo de destino.

Este objeto contiene la siguiente información:

`topicMappingNameKey`

El nombre de esta asignación de temas. *`topicMappingNameKey`*Sustitúyalo por un nombre que te ayude a identificar este mapeo de temas.

Este objeto contiene la siguiente información:

`topic`

El tema o filtro de temas que sirve de puente entre los agentes de origen y de destino.

Puede utilizar los caracteres comodín de temas MQTT + y # para transmitir mensajes en todos los temas que coincidan con un filtro de temas. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

> **Note**
>
> Para usar comodines de temas de MQTT con el agente `Pubsub` de origen, debe usar la versión 2.6.0 o posterior del componente núcleo de Greengrass.

`targetTopicPrefix`

El prefijo que se agrega al tema de destino cuando este componente transmite el mensaje.

`source`

El agente de mensajes de origen. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`target`

El agente de mensajes de destino. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`brokerUri`

(Opcional) El URI del agente MQTT local. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883. Utilice el siguiente formato y sustitúyalo por *`port`* el puerto en el que opera el broker MQTT:. `ssl://localhost:`*`port`*

Valor predeterminado: `ssl://localhost:8883`

startupTimeoutSeconds

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `BROKEN` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de archivo de configuración especifica lo siguiente:

- Transmita mensajes desde los dispositivos cliente a AWS IoT Core temas que coincidan con el filtro de `clients/+/hello/world` temas.
- Transmita los mensajes de los dispositivos cliente a los dispositivos locales publish/subscribe sobre los temas que coincidan con el `clients/+/detections` filtro de temas y añada el `events/input/` prefijo al tema de destino. El tema de destino resultante coincide con el filtro de temas `events/input/clients/+/detections`.

- Transmita los mensajes desde los dispositivos cliente a AWS IoT Core temas que coincidan con el filtro de temas y añada el `$aws/rules/StatusUpdateRule/` prefijo al `clients/+/status` tema de destino. En este ejemplo, se transmiten estos mensajes directamente a una regla AWS IoT denominada `StatusUpdateRule` para la reducción de costos mediante la ingesta básica.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.2.0 - 2.2.5

`mqttTopicMapping`

Las asignaciones de temas que desea unir. Este componente se suscribe a los mensajes del tema de origen y publica los mensajes que recibe en el tema de destino. Cada asignación de temas define el tema, el tipo de origen y el tipo de destino.

Este objeto contiene la siguiente información:

`topicMappingNameKey`

El nombre de esta asignación de temas. *`topicMappingNameKey`*Sustitúyalo por un nombre que te ayude a identificar este mapeo de temas.

Este objeto contiene la siguiente información:

`topic`

El tema o filtro de temas que sirve de puente entre los agentes de origen y de destino.

Puede utilizar los caracteres comodín de temas MQTT + y # para transmitir mensajes en todos los temas que coincidan con un filtro de temas. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

> **Note**
>
> Para usar comodines de temas de MQTT con el agente `Pubsub` de origen, debe usar la versión 2.6.0 o posterior del componente núcleo de Greengrass.

`targetTopicPrefix`

El prefijo que se agrega al tema de destino cuando este componente transmite el mensaje.

`source`

El agente de mensajes de origen. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`target`

El agente de mensajes de destino. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`brokerUri`

(Opcional) El URI del agente MQTT local. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883. Utilice el siguiente formato y sustitúyalo por *`port`* el puerto en el que opera el broker MQTT:. `ssl://localhost:`*`port`*

Valor predeterminado: `ssl://localhost:8883`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de archivo de configuración especifica lo siguiente:

- Transmita mensajes desde los dispositivos cliente a AWS IoT Core temas que coincidan con el filtro de `clients/+/hello/world` temas.
- Transmita los mensajes de los dispositivos cliente a los dispositivos locales publish/subscribe sobre los temas que coincidan con el `clients/+/detections` filtro de temas y añada el `events/input/` prefijo al tema de destino. El tema de destino resultante coincide con el filtro de temas `events/input/clients/+/detections`.

- Transmita los mensajes desde los dispositivos cliente a AWS IoT Core temas que coincidan con el filtro de temas y añada el `$aws/rules/StatusUpdateRule/` prefijo al `clients/+/status` tema de destino. En este ejemplo, se transmiten estos mensajes directamente a una regla AWS IoT denominada `StatusUpdateRule` para la reducción de costos mediante la ingesta básica.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.1.x

`mqttTopicMapping`

Las asignaciones de temas que desea unir. Este componente se suscribe a los mensajes del tema de origen y publica los mensajes que recibe en el tema de destino. Cada asignación de temas define el tema, el tipo de origen y el tipo de destino.

Este objeto contiene la siguiente información:

`topicMappingNameKey`

El nombre de esta asignación de temas. *`topicMappingNameKey`*Sustitúyalo por un nombre que te ayude a identificar este mapeo de temas.

Este objeto contiene la siguiente información:

`topic`

El tema o filtro de temas que sirve de puente entre los agentes de origen y de destino.

Si especifica el agente `LocalMqtt` o `IotCore` de origen, puede utilizar los caracteres comodín de temas MQTT + y # para retransmitir mensajes sobre todos los temas que coincidan con un filtro de temas. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

`source`

El agente de mensajes de origen. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`target`

El agente de mensajes de destino. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`brokerUri`

(Opcional) El URI del agente MQTT local. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883. Utilice el siguiente formato y sustitúyalo por *`port`* el puerto en el que opera el broker MQTT:. `ssl://localhost:`*`port`*

Valor predeterminado: `ssl://localhost:8883`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de actualización de configuración especifica la retransmisión de mensajes desde los dispositivos cliente a los AWS IoT Core `clients/MyClientDevice2/hello/world` temas `clients/MyClientDevice1/hello/world` y.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
```

```
  }
}
```

2.0.x

`mqttTopicMapping`

Las asignaciones de temas que desea unir. Este componente se suscribe a los mensajes del tema de origen y publica los mensajes que recibe en el tema de destino. Cada asignación de temas define el tema, el tipo de origen y el tipo de destino.

Este objeto contiene la siguiente información:

`topicMappingNameKey`

El nombre de esta asignación de temas. *`topicMappingNameKey`*Sustitúyalo por un nombre que ayude a identificar este mapeo de temas.

Este objeto contiene la siguiente información:

`topic`

El tema o filtro de temas que sirve de puente entre los agentes de origen y de destino.

Si especifica el agente `LocalMqtt` o `IotCore` de origen, puede utilizar los caracteres comodín de temas MQTT + y # para retransmitir mensajes sobre todos los temas que coincidan con un filtro de temas. Para obtener más información, consulte Temas MQTT en la Guía para desarrolladores de AWS IoT Core .

`source`

El agente de mensajes de origen. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del

broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

`target`

El agente de mensajes de destino. Puede elegir entre las siguientes opciones:

- `LocalMqtt`: el agente MQTT local donde se comunican los dispositivos de cliente.
- `Pubsub`— El agente de publish/subscribe mensajes local de Greengrass.
- `IotCore`— El intermediario de mensajes AWS IoT Core MQTT.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

`source` y `target` deben ser diferentes.

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de actualización de configuración especifica la retransmisión de mensajes desde los dispositivos cliente a los AWS IoT Core temas `clients/MyClientDevice1/hello/world` y`clients/MyClientDevice2/hello/world`.

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
```

```
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.3.2	Versión actualizada para la versión 2.5.0 de autenticación de dispositivos de cliente.
2.3.1	Mejoras y correcciones de errores Soluciona un problema por el que el cliente MQTT local entra en un bucle de desconexión.
2.3.0	Nuevas características Añade MQTT5 soporte para crear puentes entre fuentes MQTT locales AWS IoT Core y entre ellas.
2.2.6	Nuevas características Agrega una nueva opción de configuración `startupTimeoutSeconds` .
2.2.5	Versión actualizada para la versión 2.4.0 de autenticación de dispositivos de cliente.
2.2.4	Versión actualizada para la versión 2.3.0 de autenticación de dispositivos de cliente de Greengrass.
2.2.3	Esta versión contiene correcciones de errores y mejoras.
2.2.2	Mejoras y correcciones de errores • Ajustes de registro.
2.2.1	Mejoras y correcciones de errores Soluciona problemas que pueden provocar que el puente de MQTT no pueda suscribirse a los temas MQTT.
2.2.0	Nuevas características • Añade compatibilidad con los caracteres comodín (#y+) de los temas MQTT cuando se especifica local publish/subscribe como agente de mensajes de origen.

Versión	Cambios
	Esta característica requiere la versión 2.6.0 o posterior del componente núcleo de Greengrass. • Agrega la opción `targetTopicPrefix` , que puede especificar, para configurar el puente MQTT de forma que agregue un prefijo al tema de destino cuando retransmita un mensaje.
2.1.1	Mejoras y correcciones de errores • Soluciona problemas relacionados con la forma en que este component e administra las actualizaciones de restablecimiento de la configuración. • Reduce la frecuencia con la que el cliente MQTT se desconecta cuando los certificados rotan.
2.1.0	Nuevas características • Agrega el parámetro `brokerUri` , que le permite utilizar un puerto de agente MQTT no predeterminado.
2.0.1	Esta versión incluye correcciones de errores y mejoras.
2.0.0	Versión inicial.

Agente MQTT 3.1.1 (Moquette)

El componente agente MQTT de Moquette (`aws.greengrass.clientdevices.mqtt.Moquette`) gestiona los mensajes MQTT entre los dispositivos de cliente y un dispositivo principal de Greengrass. Este componente proporciona una versión modificada del agente MQTT de Moquette. Implemente este agente MQTT para ejecutar un agente de MQTT ligero. Para más información sobre cómo elegir un agente MQTT, consulte Elección de un agente MQTT.

Este agente implementa el protocolo MQTT 3.1.1. Incluye compatibilidad con mensajes retenidos por QoS 0, QoS 1, QoS 2 y sesiones persistentes.

> **Note**
>
> Los dispositivos de cliente son dispositivos IoT locales que se conectan a un dispositivo principal de Greengrass para enviar mensajes MQTT y datos para su procesamiento. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe poder aceptar conexiones en el puerto en el que opera el agente MQTT. Este componente ejecuta el agente MQTT en el puerto 8883 de forma predeterminada. O puede especificar un puerto diferente al configurar este componente.

  Si especifica un puerto diferente y utiliza el componente de puente MQTT para retransmitir mensajes MQTT a otros intermediarios, debe utilizar el puente de MQTT versión 2.1.0 o posterior. Configúrelo para que use el puerto en el que opera el agente MQTT.

  Si especifica un puerto diferente y utiliza el componente IP detector para administrar los puntos de conexión del agente MQTT, debe utilizar la versión 2.1.0 o posterior del detector IP. Configúrelo para que informe el puerto en el que opera el agente MQTT.
- El componente de agente MQTT de Moquette puede ejecutarse en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.3.7

En la siguiente tabla, se muestran las dependencias de la versión 2.3.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.6.0	Rígido

2.3.2 – 2.3.6

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.2 a 2.3.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.5.0	Rígido

2.3.0 and 2.3.1

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.0 y 2.3.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.4.0	Rígido

2.2.0

En la siguiente tabla, se muestran las dependencias de la versión 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.3.0	Rígido

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.0.0 <2.2.0	Rígido

2.0.0 - 2.0.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.0 a 2.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.0.0 <2.1.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`moquette`

(Opcional) La configuración del agente MQTT de Moquette que se va a utilizar. Puede configurar las opciones de configuración de Moquette en este componente. Para obtener más información, consulte los comentarios en línea en el archivo de configuración de Moquette.

Este objeto contiene la siguiente información:

`ssl_port`

(Opcional) El puerto en el que opera el agente de MQTT.

> **Note**
>
> Si especifica un puerto diferente y utiliza el componente de puente MQTT para retransmitir mensajes MQTT a otros intermediarios, debe utilizar el puente de MQTT versión 2.1.0 o posterior. Configúrelo para que use el puerto en el que opera el agente MQTT.
> Si especifica un puerto diferente y utiliza el componente IP detector para administrar los puntos de conexión del agente MQTT, debe utilizar la versión 2.1.0 o posterior del detector IP. Configúrelo para que informe el puerto en el que opera el agente MQTT.

Valor predeterminado: `8883`

`host`

(Opcional) La interfaz a la que se enlaza el agente MQTT. Por ejemplo, puede cambiar este parámetro para que el agente MQTT se vincule únicamente a una red local específica.

Predeterminado: `0.0.0.0` (enlaza con todas las interfaces de red)

startupTimeoutSeconds

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `BROKEN` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica el funcionamiento del agente de MQTT en el puerto 443.

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.3.7	Versión actualizada para la versión 2.5.0 de autenticación de dispositivos de cliente.
2.3.6	Mejoras y correcciones de errores • Corrección de errores y mejoras generales.
2.3.5	Mejoras y correcciones de errores • Se actualizó Moquette a la versión 0.17.

Versión	Cambios
2.3.4	Mejoras y correcciones de errores • Soluciona un problema por el que los clientes podían experimentar errores de sesión no válidos al enviar o recibir mensajes debido a un cliente duplicado IDs. Este problema provocó el cierre de la sesión del cliente.
2.3.3	Nuevas características Agrega una nueva opción de configuración `startupTimeoutSeconds` .
2.3.2	Versión actualizada para la versión 2.4.0 de autenticación de dispositivos de cliente.
2.3.1	Mejoras y correcciones de errores • Corrige una condición de carrera en la que los clientes podían desconectarse después de intentar volver a conectarse debido a una sesión no válida.
2.3.0	Se agregó compatibilidad con las cadenas de certificados.
2.2.0	Versión actualizada para la versión 2.2.0 de autenticación de dispositivos de cliente.
2.1.0	Mejoras y correcciones de errores • Actualiza este componente para que utilice la versión 0.16 de Moquette, que mejora el rendimiento e incluye otras mejoras. • Soluciona un problema por el que el certificado del servidor MQTT local rota con más frecuencia de lo previsto en determinadas situaciones. Para aplicar esta corrección, también debe utilizar la versión 2.1.0 o posterior del componente de autenticación del dispositivo de cliente.

Versión	Cambios
2.0.2	Mejoras y correcciones de errores • Aumenta el tamaño máximo de los mensajes MQTT de 8092 bytes a 128 kilobytes. El límite efectivo de carga útil de los mensajes en MQTT es ligeramente inferior, ya que el límite de tamaño de los mensajes incluye los encabezados de los mensajes. • Suma compatibilidad con valores enteros en el parámetro `ssl_port`.
2.0.1	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.0	Versión inicial.

Agente MQTT 5 (EMQX)

El componente agente MQTT de EMQX (`aws.greengrass.clientdevices.mqtt.EMQX`) gestiona los mensajes MQTT entre los dispositivos de cliente y un dispositivo principal de Greengrass. Este componente proporciona una versión modificada del agente EMQX MQTT 5.0. Implemente el agente MQTT para usar las características de MQTT 5 en la comunicación entre los dispositivos de cliente y el dispositivo principal. Para más información sobre cómo elegir un agente MQTT, consulte Elección de un agente MQTT.

Este agente implementa el protocolo MQTT 5.0. Incluye compatibilidad con intervalos de expiración de sesiones y mensajes, propiedades de usuario, suscripciones compartidas, alias de temas y más. MQTT 5 es compatible con versiones anteriores de MQTT 3.1.1, por lo que si ejecuta el agente Moquette MQTT 3.1.1, puede reemplazarlo con el agente EMQX MQTT 5 y los dispositivos de cliente pueden continuar conectándose y funcionando normalmente.

> **Note**
>
> Los dispositivos de cliente son dispositivos IoT locales que se conectan a un dispositivo principal de Greengrass para enviar mensajes MQTT y datos para su procesamiento. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Licencias
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.0.x
- 1.2.x
- 1.1.x
- 1.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe poder aceptar conexiones en el puerto en el que opera el agente MQTT. Este componente ejecuta el agente MQTT en el puerto 8883 de forma predeterminada. O puede especificar un puerto diferente al configurar este componente.

  Si especifica un puerto diferente y utiliza el componente de puente MQTT para retransmitir mensajes MQTT a otros intermediarios, debe utilizar el puente de MQTT versión 2.1.0 o posterior. Configúrelo para que use el puerto en el que opera el agente MQTT.

  Si especifica un puerto diferente y utiliza el componente IP detector para administrar los puntos de conexión del agente MQTT, debe utilizar la versión 2.1.0 o posterior del detector IP. Configúrelo para que informe el puerto en el que opera el agente MQTT.
- En los dispositivos principales de Linux, Docker se instala y configura en el dispositivo principal:
  - Versión 1.9.1 o posterior de Docker Engine instalada en el dispositivo principal de Greengrass. La versión 20.10 es la última versión que se ha comprobado que funciona con el software AWS IoT Greengrass Core. Debe instalar Docker directamente en el dispositivo principal antes de implementar componentes que ejecuten contenedores de Docker.
  - El daemon de Docker se inició y se ejecutó en el dispositivo principal antes de implementar este componente.
  - El usuario del sistema que ejecuta este componente debe tener permisos de administrador o raíz. Como alternativa, puede ejecutar este componente como usuario del sistema en el grupo `docker` y configurar la opción `requiresPrivileges` de este componente `false` para ejecutar el agente MQTT de EQMX sin privilegios.
- Se admite la ejecución del componente agente MQTT de EMQX en una VPC.
- No se admite el componente agente MQTT de EMQX en la plataforma `armv7`.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia.

También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.0.2 – 2.0.3

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.2 y 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.6.0	Flexible

2.0.1

En la siguiente tabla, se muestran las dependencias de la versión 2.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.6.0	Rígido

2.0.0

En la siguiente tabla, se muestran las dependencias de la versión 2.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.5.0	Rígido

1.2.2 – 1.2.3

En la siguiente tabla, se muestran las dependencias de las versiones 1.2.2 a 1.2.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.5.0	Rígido

1.2.0 and 1.2.1

En la siguiente tabla, se muestran las dependencias de las versiones 1.2.0 y 1.2.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.4.0	Rígido

1.0.0 and 1.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.0 y 1.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Autenticación del dispositivo de cliente	>=2.2.0 <2.3.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

2.0.0 - 2.0.3

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

> **Important**
>
> Si utiliza la versión 2 del componente agente MQTT 5 (EMQX), debe actualizar el archivo de configuración. Los archivos de configuración de la versión 1 no funcionan con la versión 2.

emqxConfig

(Opcional) La configuración del agente EMQX MQTT que se va a utilizar. Puede configurar las opciones de configuración de EMQX en este componente.

Cuando se utiliza el agente EMQX, Greengrass utiliza una configuración predeterminada. Esta configuración se utiliza a menos que la modifique mediante este campo.

La modificación de los siguientes ajustes de configuración provoca el reinicio del componente agente EMQX. Se aplican otros cambios de configuración sin reiniciar el componente.

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX` permite configurar opciones sensibles a la seguridad. Estas incluyen la configuración de TLS, la autenticación y los proveedores de autorización. Recomendamos la configuración predeterminada que utiliza la autenticación TLS mutua y el proveedor de autenticación de dispositivos de cliente Greengrass.

Example Ejemplo: configuración predeterminada

El siguiente ejemplo muestra los valores predeterminados establecidos para el agente MQTT 5 (EMQX). Puede anular estos parámetros mediante el ajuste de configuración `emqxConfig`.

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
```

```
  },
  "node": {
    "cookie": "<placeholder>"
  },
  "listeners": {
     "ssl": {
       "default": {
         "ssl_options": {
           "keyfile": "{work:path}\\data\\key.pem",
           "certfile": "{work:path}\\data\\cert.pem",
           "cacertfile": null,
           "verify": "verify_peer",
           "versions": ["tlsv1.3", "tlsv1.2"],
           "fail_if_no_peer_cert": true
         }
       }
     },
     "tcp": {
       "default": {
         "enabled": false
       }
     },
     "ws": {
       "default": {
         "enabled": false
       }
     },
     "wss": {
       "default": {
         "enabled": false
       }
     }
  },
  "plugins": {
    "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
    "install_dir": "plugins"
  }
}
```

authMode

(Opcional) Establece el proveedor de autorización del agente. Puede ser uno de los siguientes valores:

- `enabled`: (predeterminado) utilice el proveedor de autenticación y autorización de Greengrass.
- `bypass_on_failure`: utilice el proveedor de autenticación de Greengrass y, a continuación, utilice los demás proveedores de autenticación de la cadena de proveedores de EMQX si Greengrass deniega la autenticación o la autorización.
- `bypass`: el proveedor de Greengrass está deshabilitado. La cadena de proveedores de EMQX gestiona la autenticación y la autorización.

`requiresPrivilege`

(Opcional) En los dispositivos principales de Linux, puede especificar que se ejecute el agente de MQTT EMQX sin privilegios de administrador o raíz. Si establece esta opción en `false`, el usuario del sistema que ejecute este componente debe ser miembro del grupo `docker`.

Valor predeterminado: `true`

`startupTimeoutSeconds`

(Opcional) El tiempo máximo en segundos para que se inicie el agente de MQTT EMQX. El estado del componente cambia a `BROKEN` si supera este tiempo de espera.

Valor predeterminado: `90`

`ipcTimeoutSeconds`

(Opcional) El tiempo máximo en segundos que tarda el componente en esperar a que el núcleo de Greengrass responda a las solicitudes de comunicación entre procesos (IPC). Aumente este número si este componente informa de errores de tiempo de espera al comprobar si un dispositivo de cliente está autorizado.

Valor predeterminado: `5`

`crtLogLevel`

(Opcional) El nivel de registro de la biblioteca AWS Common Runtime (CRT).

El valor predeterminado es el nivel de registro del agente de MQTT EMQX (`log.level` en `emqx`).

`restartIdentifier`

(Opcional) Configure esta opción para reiniciar el agente de MQTT EMQX. Cuando este valor de configuración cambia, este componente reinicia el agente de MQTT. Puede utilizar esta opción para forzar la desconexión de los dispositivos de cliente.

`dockerOptions`

(Opcional) Configure esta opción solo en los sistemas operativos Linux para agregar parámetros a la línea de comandos de Docker. Por ejemplo, para asignar puertos adicionales, utilice el parámetro `-p` de Docker:

```
"-p 1883:1883"
```

Example Ejemplo: actualizar un archivo de configuración de la versión 1.x a la versión 2.x

El siguiente ejemplo muestra los cambios necesarios para actualizar un archivo de configuración de la versión 1.x a la versión 2.x.

El archivo de configuración de la versión 1.x:

```
{
    "emqx": {
        "listener.ssl.external": "443",
        "listener.ssl.external.max_connections": "1024000",
        "listener.ssl.external.max_conn_rate": "500",
        "listener.ssl.external.rate_limit": "50KB,5s",
        "listener.ssl.external.handshake_timeout": "15s",
        "log.level": "warning"
    },
    "mergeConfigurationFiles": {
        "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
    }
}
```

El archivo de configuración equivalente para la versión 2:

```
{
    "emqxConfig": {
        "listeners": {
            "ssl": {
                "default": {
                   "bind": "8883",
                   "max_connections": "1024000",
                   "max_conn_rate": "500",
```

```
                    "ssl_options": {
                         "handshake_timeout": "15s"
                    }
                }
            }
        },
        "log": {
            "console": {
              "enable": true,
              "level": "warning"
            }
        }
    },
    "authMode": "enabled"
}
```

No hay ningún equivalente a la entrada `listener.ssl.external.rate_limit` de configuración. La opción `use_greengrass_managed_certificates` de configuración se ha eliminado.

Example Ejemplo: establecer un puerto nuevo para el agente

El siguiente ejemplo cambia el puerto en el que opera el agente de MQTT del 8883 predeterminado al puerto 1234. Si utiliza Linux, incluya el campo `dockerOptions`.

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example Ejemplo: ajustar el nivel de registro del agente de MQTT

El siguiente ejemplo cambia el nivel de registro del agente de MQTT a `debug`. Puede elegir entre los siguientes niveles de registro:

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

El nivel de registro predeterminado es `warning`.

```
{
  "emqxConfig": {
    "log": {
      "console": {
         "level": "debug"
      }
    }
  }
}
```

Example Ejemplo: habilitar el panel de EMQX

El siguiente ejemplo habilita el panel de EMQX para que pueda monitorear y administrar su agente. Si utiliza Linux, incluya el campo `dockerOptions`.

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`emqx`

(Opcional) La configuración del agente EMQX MQTT que se va a utilizar. Puede configurar un subconjunto de opciones de configuración de EMQX en este componente.

Este objeto contiene la siguiente información:

`listener.ssl.external`

(Opcional) El puerto en el que opera el agente de MQTT.

> **Note**
>
> Si especifica un puerto diferente y utiliza el componente de puente MQTT para retransmitir mensajes MQTT a otros intermediarios, debe utilizar el puente de MQTT versión 2.1.0 o posterior. Configúrelo para que use el puerto en el que opera el agente MQTT.
> Si especifica un puerto diferente y utiliza el componente IP detector para administrar los puntos de conexión del agente MQTT, debe utilizar la versión 2.1.0 o posterior del detector IP. Configúrelo para que informe el puerto en el que opera el agente MQTT.

Valor predeterminado: `8883`

`listener.ssl.external.max_connections`

(Opcional) el número máximo de conexiones simultáneas que admite el agente MQTT.

Valor predeterminado: `1024000`

`listener.ssl.external.max_conn_rate`

(Opcional) El número máximo de conexiones nuevas por segundo que puede recibir el agente de MQTT.

Valor predeterminado: `500`

`listener.ssl.external.rate_limit`

(Opcional) El límite de ancho de banda para todas las conexiones al agente de MQTT. Especifique el ancho de banda y la duración de ese ancho de banda separados por una coma (`,`) en el siguiente formato: `bandwidth,duration`. Por ejemplo, puede especificar `50KB,5s` para limitar el agente de MQTT a 50 kilobytes (KB) de datos cada 5 segundos.

`listener.ssl.external.handshake_timeout`

(Opcional) El tiempo que espera el agente de MQTT para terminar de autenticar una nueva conexión.

Valor predeterminado: `15s`

`mqtt.max_packet_size`

(Opcional) El tamaño máximo de un mensaje MQTT.

Predeterminado: `268435455` (256 MB menos 1)

`log.level`

(Opcional) El nivel de registro del agente de MQTT. Puede elegir entre las siguientes opciones:

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

El nivel de registro predeterminado es `warning`.

`requiresPrivilege`

(Opcional) En los dispositivos principales de Linux, puede especificar que se ejecute el agente de MQTT EMQX sin privilegios de administrador o raíz. Si establece esta opción en `false`, el usuario del sistema que ejecute este componente debe ser miembro del grupo `docker`.

Valor predeterminado: `true`

`startupTimeoutSeconds`

(Opcional) El tiempo máximo en segundos para que se inicie el agente de MQTT EMQX. El estado del componente cambia a `BROKEN` si supera este tiempo de espera.

Valor predeterminado: `90`

`ipcTimeoutSeconds`

(Opcional) El tiempo máximo en segundos que tarda el componente en esperar a que el núcleo de Greengrass responda a las solicitudes de comunicación entre procesos (IPC). Aumente este número si este componente informa de errores de tiempo de espera al comprobar si un dispositivo de cliente está autorizado.

Valor predeterminado: `5`

`crtLogLevel`

(Opcional) El nivel de registro de la biblioteca AWS Common Runtime (CRT).

El valor predeterminado es el nivel de registro del agente de MQTT EMQX (`log.level` en `emqx`).

`restartIdentifier`

(Opcional) Configure esta opción para reiniciar el agente de MQTT EMQX. Cuando este valor de configuración cambia, este componente reinicia el agente de MQTT. Puede utilizar esta opción para forzar la desconexión de los dispositivos de cliente.

`dockerOptions`

(Opcional) Configure esta opción solo en los sistemas operativos Linux para agregar parámetros a la línea de comandos de Docker. Por ejemplo, para asignar puertos adicionales, utilice el parámetro `-p` de Docker:

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

(Opcional) Configure esta opción para agregar o anular los valores predeterminados de los archivos de configuración de EMQX especificados. Para obtener información sobre los archivos de configuración y sus formatos, consulte Configuración en la documentación de EMQX 4.0. Los valores que especifique se adjuntan al archivo de configuración.

El siguiente ejemplo actualiza el archivo `etc/emqx.conf`.

```
"mergeConfigurationFiles": {
    "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"
},
```

Además de los archivos de configuración compatibles con EMQX, Greengrass admite un archivo que configura el complemento de autenticación de Greengrass para EMQX llamado `etc/plugins/aws_greengrass_emqx_auth.conf`. Hay dos opciones compatibles: `auth_mode` y `use_greengrass_managed_certificates`. Para usar otro proveedor de autenticación, defina la opción `auth_mode` en una de las siguientes opciones:

- `enabled`: (predeterminado) utilice el proveedor de autenticación y autorización de Greengrass.
- `bypass_on_failure`: utilice el proveedor de autenticación de Greengrass y, a continuación, utilice los demás proveedores de autenticación de la cadena de proveedores de EMQX si Greengrass deniega la autenticación o la autorización.
- `bypass`: el proveedor de Greengrass está deshabilitado. La cadena de proveedores de EMQX gestiona la autenticación y la autorización.

Si `use_greengrass_managed_certificates` es `true`, esta opción indica que Greengrass administra los certificados TLS del agente. Si `false`, indica que proporciona los certificados a través de otro origen.

En el siguiente ejemplo, se actualizan los valores predeterminados del archivo `etc/plugins/aws_greengrass_emqx_auth.conf` de configuración.

```
"mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
  },
```

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX` permite configurar opciones sensibles a la seguridad. Estas incluyen la configuración de TLS, la autenticación y los proveedores de autorización. La configuración recomendada es la configuración predeterminada que utiliza autenticación TLS mutua y el proveedor de autenticación de dispositivo de cliente de Greengrass.

`replaceConfigurationFiles`

(Opcional) Configure esta opción para reemplazar los archivos de configuración de EMQX especificados. Los valores que especifique reemplazan todo el archivo de configuración existente. No puede especificar el archivo `etc/emqx.conf` en esta sección. Debe usar `mergeConfigurationFile` para modificar `etc/emqx.conf`.

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica el funcionamiento del agente de MQTT en el puerto 443.

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
  "startupTimeoutSeconds": "90",
  "ipcTimeoutSeconds": "5"
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -
  Tail 10 -Wait
  ```

Licencias

En los sistemas operativos Windows, este software incluye código distribuido según los términos de licencia del software de Microsoft: Microsoft Visual Studio Community 2022. Al descargar este software, acepta los términos de licencia del código.

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

v2.x

Versión	Cambios
2.0.3	Mejoras y correcciones de errores • Soluciona un problema que impedía a EMQX iniciar en Windows si la ruta contenía espacios.
2.0.2	Mejoras y correcciones de errores • Soluciona un problema que iniciaba EMQX antes de que el componente de autenticación del dispositivo cliente estuviera listo.

Versión	Cambios
2.0.1	Versión actualizada para la versión 2.5.0 de autenticación de dispositivos de cliente.
2.0.0	Esta versión del agente MQTT 5 (EMQX) espera parámetros de configuración diferentes a los de la versión 1.x. Si utiliza una configuración no predeterminada para la versión 1.x, debe actualizar la configuración del componente para la versión 2.x. Para obtener más información, consulte Configuración. Nuevas características • Se actualiza el agente de MQTT a EMQX 5.1.1. • Permite cambios de configuración del agente sin reiniciar el componente. Actualizaciones • Agrega un nuevo campo de configuración `emqxConfig` que reemplaza los campos de configuración `emqx`, `mergeConfigurationFiles` y `replaceConfigurationFiles`.

v1.x

Versión	Cambios
1.2.3	Mejoras y correcciones de errores • Soluciona un problema por el que los clientes no podían interactuar con EMQX después de autenticarse previamente al desconectar y volver a autenticar el cliente.
1.2.2	Versión actualizada para la versión 2.4.0 de autenticación de dispositivos de cliente.
1.2.1	Mejoras y correcciones de errores • Soluciona un problema por el que el componente no se iniciaba en Windows si Visual C++ Redistributable aún no estaba presente.

Versión	Cambios
	• Actualiza EMQX a la versión 4.4.14.
1.2.0	Se agregó compatibilidad con las cadenas de certificados.
1.1.0	Nuevas características • Suma compatibilidad con las configuraciones de EMQX, incluidas las opciones de agente y los complementos. Mejoras y correcciones de errores • Actualiza EMQX a la versión 4.4.9.
1.0.1	Corrige un problema durante el protocolo de enlace TLS que provocaba que algunos clientes MQTT no pudieran conectarse.
1.0.0	Versión inicial.

Emisor de telemetría del núcleo

El componente emisor de telemetría nuclear (`aws.greengrass.telemetry.NucleusEmitter`) recopila datos de telemetría de salud del sistema y los publica continuamente sobre un tema local y otro sobre el MQTT. AWS IoT Core Este componente le permite recopilar la telemetría del sistema en tiempo real en sus dispositivos principales de Greengrass. Para obtener información sobre el agente de telemetría de Greengrass que publica los datos de telemetría del sistema en Amazon, consulte. EventBridge Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass

De forma predeterminada, el componente emisor de telemetría del núcleo publica los datos de telemetría cada 60 segundos en el siguiente tema local. publish/subscribe

```
$local/greengrass/telemetry
```

De forma predeterminada, el componente emisor de telemetría nuclear no se publica en ningún tema de MQTT. AWS IoT Core Puede configurar este componente para que se publique en un tema de AWS IoT Core MQTT al implementarlo. El uso de un tema de MQTT para publicar datos en el Nube de AWS está sujeto a AWS IoT Core los precios.

AWS IoT Greengrass proporciona varios componentes comunitarios para ayudarlo a analizar y visualizar los datos de telemetría localmente en su dispositivo principal mediante InfluxDB y Grafana. Estos componentes utilizan datos de telemetría del componente emisor de núcleo. Para obtener más información, consulte el archivo README para el componente publicador de InfluxDB.

Temas

- Versiones
- Tipo
- Sistema operativo
- Dependencias
- Configuración
- Datos de salida
- De uso
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 1.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

1.0.12

La siguiente tabla muestra las dependencias de la versión 1.0.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.17.0	Flexible

1.0.11

La siguiente tabla muestra las dependencias de la versión 1.0.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.16.0	Rígido

1.0.10

LEn la siguiente tabla, se muestran las dependencias de la versión 1.0.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 =2.4.0 <2.15.0	Rígido

1.0.9

En la siguiente tabla, se muestran las dependencias de la versión 1.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.14.0	Rígido

1.0.8

En la siguiente tabla, se muestran las dependencias de la versión 1.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.13.0	Rígido

1.0.7

En la siguiente tabla, se muestran las dependencias de la versión 1.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.12.0	Rígido

1.0.6

En la siguiente tabla, se muestran las dependencias de la versión 1.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.11.0	Rígido

1.0.5

En la siguiente tabla, se muestran las dependencias de la versión 1.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.10.0	Rígido

1.0.4

En la siguiente tabla, se muestran las dependencias de la versión 1.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.9.0	Rígido

1.0.3

En la siguiente tabla, se muestran las dependencias de la versión 1.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.8.0	Rígido

1.0.2

En la siguiente tabla, se muestran las dependencias de la versión 1.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.7.0	Rígido

1.0.1

En la siguiente tabla, se muestran las dependencias de la versión 1.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.6.0	Rígido

1.0.0

En la siguiente tabla, se muestran las dependencias de la versión 1.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.4.0 <2.5.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`pubSubPublish`

(Opcional) Define si se deben publicar los datos de telemetría en el tema `$local/greengrass/telemetry`. Los valores admitidos son `true` y `false`.

Valor predeterminado: `true`

`mqttTopic`

(Opcional) El tema de AWS IoT Core MQTT en el que este componente publica datos de telemetría.

Establezca este valor en el tema AWS IoT Core MQTT en el que desee publicar los datos de telemetría. Cuando este valor está vacío, el emisor del núcleo no publica los datos de telemetría en la Nube de AWS.

> **Note**
>
> El uso de un tema de MQTT para publicar datos en el Nube de AWS está sujeto a los precios.AWS IoT Core

Valor predeterminado: `""`

`telemetryPublishIntervalMs`

(Opcional) Cantidad de tiempo (en milisegundos) que el componente publica los datos de telemetría. Si establece este valor por debajo del valor mínimo admitido, el componente utilizará el valor mínimo en su lugar.

> **Note**
>
> Los intervalos de publicación más bajos dan como resultado un mayor uso de la CPU en el dispositivo principal. Le recomendamos empezar con el intervalo de publicación predeterminado y ajustarlo en función del uso de la CPU del dispositivo.

Mínimo: `500`

Valor predeterminado: `60000`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo muestra un ejemplo de actualización de combinación de configuraciones que permite publicar datos de telemetría cada 5 segundos en el `$local/greengrass/telemetry` tema y en el tema MQTT. `greengrass/myTelemetry` AWS IoT Core

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

Datos de salida

Este componente publica las métricas de telemetría como una matriz JSON en el tema siguiente.

Tema local: `$local/greengrass/telemetry`

Si lo desea, puede optar por publicar también las métricas de telemetría en un tema de MQTT. AWS IoT Core Para obtener más información, consulte Temas de MQTT en la Guía para desarrolladores de AWS IoT Core .

Example Datos de ejemplo


```
[
  {
    "A": "Average",
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
    "A": "Count",
    "N": "TotalNumberOfFDs",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Count",
    "V": 7316
  },
  {
    "A": "Count",
    "N": "SystemMemUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Megabytes",
    "V": 10098
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStarting",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsInstalled",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
```

```
    "A": "Count",
    "N": "NumberOfComponentsStateless",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStopping",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsBroken",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsRunning",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 7
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsErrored",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsNew",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
```

```
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsFinished",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 2
  }
]
```

La matriz de salida contiene una lista de métricas que tienen las siguientes propiedades:

`A`

El tipo de agregación de las métricas.

Para la métrica `CpuUsage`, esta propiedad se establece en `Average` porque el valor publicado de la métrica es la cantidad media de uso de la CPU desde el último evento de publicación.

Para el resto de las métricas, el emisor del núcleo no agrega el valor de la métrica y esta propiedad se establece en `Count`.

`N`

El nombre de la métrica.

`NS`

Espacio de nombres de la métrica.

`TS`

Marca temporal del momento en que se recopilaron los datos.

`U`

La unidad del valor métrico.

`V`

El valor de la métrica de .

El emisor del núcleo publica las siguientes métricas:

Name	Description (Descripción)
System (Sistema)	
`SystemMemUsage`	La cantidad de memoria que utilizan actualmente todas las aplicaciones del dispositivo principal de Greengrass, incluido el sistema operativo.
`CpuUsage`	La cantidad de CPU que utilizan actualmente todas las aplicaciones del dispositivo principal de Greengrass, incluido el sistema operativo.
`TotalNumberOfFDs`	El número de descriptores de archivos almacenados por el sistema operativo del dispositivo principal de Greengrass. Un descriptor de archivo identifica exclusivamente un archivo abierto.
Núcleo de Greengrass	
`NumberOfComponentsRunning`	El número de componentes que se ejecutan en el dispositivo principal de Greengrass.
`NumberOfComponentsErrored`	El número de componentes que están en estado de error en el dispositivo principal de Greengrass.
`NumberOfComponentsInstalled`	El número de componentes instalados en el dispositivo principal de Greengrass.

Name	Description (Descripción)
`NumberOfComponentsStarting`	El número de componentes que se inician en el dispositivo principal de Greengrass.
`NumberOfComponentsNew`	El número de component es nuevos en el dispositivo principal de Greengrass.
`NumberOfComponentsStopping`	El número de componentes que se detienen en el dispositi vo principal de Greengrass.
`NumberOfComponentsFinished`	El número de componentes terminados en el dispositivo principal de Greengrass.
`NumberOfComponentsBroken`	El número de componentes dañados en el dispositivo principal de Greengrass.
`NumberOfComponentsStateless`	El número de componentes sin estado en el dispositivo principal de Greengrass.

De uso

Para utilizar los datos de telemetría del estado del sistema, puede crear componentes personalizados que se adapten a los temas en los que el emisor del núcleo publica los datos de telemetría y que reaccionen a esos datos según sea necesario. Como el componente emisor del núcleo ofrece la opción de publicar datos de telemetría en un tema local, puede suscribirse a ese tema y utilizar los datos publicados para actuar de forma local en su dispositivo principal. De este modo, el dispositivo principal puede reaccionar a los datos de telemetría incluso cuando su conectividad a la nube sea limitada.

Por ejemplo, puede configurar un componente que escuche el tema `$local/greengrass/telemetry` en busca de datos de telemetría y envíe los datos al componente del administrador

de flujos para transmitir sus datos a la Nube de AWS. Para obtener más información sobre cómo crear este componente, consulte Publicar/suscribir mensajes locales y Creación de componentes personalizados que usen el administrador de flujos.

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la AWS IoT Greengrass carpeta raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.0.12	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.

Versión	Cambios
1.0.11	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
1.0.10	Versión actualizada para la versión 2.14.0 de Greengrass nucleus.
1.0.9	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
1.0.8	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
1.0.7	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
1.0.6	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
1.0.5	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
1.0.4	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
1.0.3	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
1.0.2	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
1.0.1	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
1.0.0	Versión inicial.

Proveedor PKCS#11

El componente proveedor PKCS#11 (`aws.greengrass.crypto.Pkcs11Provider`) le permite configurar el software AWS IoT Greengrass Core para utilizar un módulo de seguridad de hardware

(HSM) a través de la interfaz PKCS#11. Esta característica le permite almacenar de forma segura los archivos de certificados y claves privadas para que no queden expuestos ni duplicados en el software. Para obtener más información, consulte Integración de la seguridad de hardware.

Para aprovisionar un dispositivo principal de Greengrass que almacene su certificado y su clave privada en un HSM, debe especificar este componente como un complemento de aprovisionamiento al instalar el software Core. AWS IoT Greengrass Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.

AWS IoT Greengrass proporciona este componente como un archivo JAR que puede descargar para especificarlo como complemento de aprovisionamiento durante la instalación. Puede descargar la última versión del archivo JAR del componente en la siguiente URL: https://d2s8p88vqu9w66.cloudfront. net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-latest.jar.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- Un módulo de seguridad de hardware que admite el esquema de firmas PKCS#1 versión 1.5 y claves RSA con un tamaño de clave RSA-2048 (o mayor) o claves ECC.

  > **Note**
  >
  > Para utilizar un módulo de seguridad de hardware con claves ECC, debe utilizar la versión del núcleo de Greengrass 2.5.6 o posterior.
  > Para usar un módulo de seguridad de hardware y el administrador de secretos, debe usar un módulo de seguridad de hardware con claves RSA.

- Una biblioteca de proveedores de PKCS #11 que el software AWS IoT Greengrass principal puede cargar en tiempo de ejecución (mediante libdl) para invocar las funciones de PKCS #11. La biblioteca de proveedores PKCS#11 debe implementar las siguientes operaciones de la API de PKCS#11:
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`

  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- El módulo de hardware debe resolverlo la etiqueta de ranura, tal y como se define en la especificación de PKCS#11.
- Debe almacenar la clave privada y el certificado en el HSM, en la misma ranura, y deben usar la misma etiqueta de objeto e ID de objeto, si el HSM admite el objeto. IDs
- El certificado y la clave privada debe poder resolverla con etiquetas de objeto.
- La clave privada debe tener los siguientes permisos:
  - `sign`
  - `decrypt`
- (Opcional) Para usar el componente administrador de secretos, debe usar la versión 2.1.0 o posterior y la clave privada debe tener los siguientes permisos:
  - `unwrap`
  - `wrap`
- (Opcional) Si utiliza la TPM2 biblioteca y ejecuta el núcleo de Greengrass como servicio, debe proporcionar una variable de entorno con la ubicación del almacén PKCS #11. El siguiente ejemplo es un archivo de servicio de systemd con la variable de entorno requerida:

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

[Install]
WantedBy=multi-user.target
```

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.0.11

En la siguiente tabla, se muestran las dependencias de la versión 2.0.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.17.0	Flexible

2.0.10

En la siguiente tabla, se muestran las dependencias de la versión 2.0.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 =2.5.3 <2.16.0	Flexible

2.0.9

En la siguiente tabla, se muestran las dependencias de la versión 2.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 =2.5.3 <2.15.0	Flexible

2.0.8

En la siguiente tabla, se muestran las dependencias de la versión 2.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.14.0	Flexible

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.13.0	Flexible

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.12.0	Flexible

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.11.0	Flexible

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.10.0	Flexible

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.9.0	Flexible

2.0.2

En la siguiente tabla, se muestran las dependencias de la versión 2.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.8.0	Flexible

2.0.1

En la siguiente tabla, se muestran las dependencias de la versión 2.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.7.0	Flexible

2.0.0

En la siguiente tabla, se muestran las dependencias de la versión 2.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.3 <2.6.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`name`

Un nombre para la configuración de PKCS#11.

`library`

La ruta absoluta del archivo a la biblioteca de la implementación de PKCS #11 que el software Core puede cargar con libdl. AWS IoT Greengrass

`slot`

El ID de la ranura que contiene la clave privada y el certificado del dispositivo. Este valor es diferente del índice o la etiqueta de la ranura.

`userPin`

El PIN del usuario que se utiliza para acceder a la ranura.

Example Ejemplo: actualización de la combinación de configuraciones

```
{
```

```
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.0.11	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.
2.0.10	Versión actualizada para el lanzamiento de la versión 2.15.0 de Greengrass nucleus.
2.0.9	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.0.3	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.0.2	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.0.1	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.0.0	Versión inicial.

Administrador de secretos

El componente administrador de secretos (`aws.greengrass.SecretManager`) implementa secretos desde AWS Secrets Manager a los dispositivos principales de Greengrass. Utilice

este componente para utilizar de forma segura las credenciales, como las contraseñas, en los componentes personalizados de sus dispositivos principales de Greengrass. Para obtener más información acerca de Secrets Manager, consulte ¿Qué es AWS Secrets Manager? en la Guía del usuario de AWS Secrets Manager .

Para acceder a los secretos de este componente en sus componentes personalizados de Greengrass, utilice la GetSecretValueoperación de. SDK para dispositivos con AWS IoT Para obtener más información, consulte Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core y Recupere valores secretos.

Este componente cifra los secretos del dispositivo principal para mantener sus credenciales y contraseñas seguras hasta que necesite usarlas. Utiliza la clave privada del dispositivo principal para cifrar y descifrar los secretos.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El rol del dispositivo de Greengrass debe permitir la acción `secretsmanager:GetSecretValue`, tal como se muestra en la siguiente política de IAM de ejemplo.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
      "arn:aws:secretsmanager:us-east-1:123456789012:secret:MySecret"
      ]
    }
```

```
    ]
}
```

> **Note**
>
> Si utiliza una AWS Key Management Service clave gestionada por el cliente para cifrar los secretos, la función del dispositivo también debe permitir la acción. `kms:Decrypt`

Para obtener más información sobre las políticas de IAM para el administrador de secretos, consulte los siguientes temas en la Guía del usuario de AWS Secrets Manager :

- Autenticación y control de acceso para AWS Secrets Manager
- Acciones, recursos y claves de contexto que puede utilizar en una política de IAM o política de secretos para AWS Secrets Manager

- Los componentes personalizados deben definir una política de autorización que permita a `aws.greengrass#GetSecretValue` obtener los secretos que se almacenan con este componente. En esta política de autorización, puede restringir el acceso de los componentes a secretos específicos. Para obtener más información, consulte Autorización de IPC del administrador de secretos.
- (Opcional) Si almacena la clave privada y el certificado del dispositivo principal en un módulo de seguridad de hardware (HSM), el HSM debe admitir las claves RSA, la clave privada debe tener el permiso `unwrap` y la clave pública debe tener el permiso `wrap`.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`secretsmanager. region.amazonaws.com`	443	Sí	Descargue los secretos del dispositivo principal.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.2.7

En la siguiente tabla, se muestran las dependencias de la versión 2.2.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.17.0	Flexible

2.2.6

En la siguiente tabla, se muestran las dependencias de la versión 2.2.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 =2.5.0 <2.16.0	Flexible

2.2.2 – 2.2.5

La siguiente tabla muestra las dependencias de las versiones 2.2.2 a 2.2.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.15.0	Flexible

2.2.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.13.0 <2.14.0	Flexible

2.1.7 – 2.1.8

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.7 y 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.13.0	Flexible

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.12.0	Flexible

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.11.0	Flexible

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.10.0	Flexible

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.9.0	Flexible

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.8.0	Flexible

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.7.0	Flexible

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.6.0	Flexible

2.0.9

En la siguiente tabla, se muestran las dependencias de la versión 2.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

2.0.8

En la siguiente tabla, se muestran las dependencias de la versión 2.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

2.0.4 and 2.0.5

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.4 y 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`periodicRefreshIntervalMin` (opcional)

El intervalo en minutos en el que este componente sincroniza los secretos configurados en el dispositivo principal con los valores secretos más recientes del servicio. AWS Secrets Manager Si este intervalo no está configurado, el administrador de secretos no actualizará los secretos configurados periódicamente.

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef"
    }
  ],
  "periodicRefreshIntervalMin" : 60
}
```

`cloudSecrets`

Una lista de los secretos del administrador de secretos para implementarlos en el dispositivo principal. Puede especificar etiquetas para definir qué versiones de cada secreto se van a implementar. Si no especifica ninguna versión, este componente implementa la versión con la etiqueta `AWSCURRENT` provisional adjunta. Para obtener más información, consulte Etiquetas provisionales en la Guía del usuario de AWS Secrets Manager .

El componente de administrador de secretos almacena en caché los secretos de forma local. Si el valor secreto cambia en el administrador de secretos, este componente no recupera automáticamente el nuevo valor. Para actualizar la copia local, asigne una nueva etiqueta al secreto y configure este componente para recuperar el secreto identificado por la nueva etiqueta.

Cada objeto contiene la siguiente información:

`arn`

El ARN del secreto a implementar. El ARN del secreto puede ser un ARN completo o parcial. Para un ARN, le recomendamos que especifique un ARN completo en lugar de un ARN parcial. Para obtener más información, consulte Búsqueda de un secreto a partir de un ARN parcial. El siguiente es un ejemplo de un ARN completo y un ARN parcial:

- ARN completo: `arn:aws:secretsmanager:us-east-2:111122223333:secret:`*`SecretName`*`-abcdef`
- ARN parcial: `arn:aws:secretsmanager:us-east-2:111122223333:secret:`*`SecretName`*

`labels`

(Opcional) Una lista de etiquetas para identificar las versiones del secreto que se van a implementar en el dispositivo principal.

Cada etiqueta debe ser una cadena.

Example Ejemplo: actualización de la combinación de configuraciones

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
```

```
    }
  ]
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.2.7	Versión actualizada para la versión 2.16.0 de Greengrass nucleus.

Versión	Cambios
2.2.6	Mejoras y correcciones de errores Soluciona un problema que impedía que el administrador de secretos obtuviera un secreto debido a que un módulo de plataforma confiable era lento o no respondía. Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.2.5	Mejoras y correcciones de errores Soluciona un problema por el que un secreto no se recupera de la caché local Nube de AWS si no está presente en la memoria caché local.
2.2.4	Mejoras y correcciones de errores Reduce la frecuencia de las escrituras en el almacén de secretos local. El administrador de secretos ahora escribe en la tienda local solo cuando se actualizan los secretos.
2.2.3	Mejoras y correcciones de errores Soluciona un problema que impedía al administrador de secretos borrar los secretos almacenados de forma local cuando el dispositivo principal está desconectado, y el servicio de seguridad del dispositivo (como un HSM) no está disponible.
2.2.2	Mejoras y correcciones de errores Soluciona un problema que impedía que el administrador de secretos descargara los secretos configurados con ARN parciales.
2.2.1	Mejoras y correcciones de errores Admite el administrador de secretos en las versiones 2.5.0 y posteriores del núcleo.

Versión	Cambios
2.2.0	Nuevas características Suma compatibilidad para la actualización periódica de los secretos configurados mediante una nueva clave de configuración de component es. Suma compatibilidad con un nuevo parámetro de solicitud en la solicitud `GetSecretValue` de IPC para actualizar los secretos por solicitud
2.1.8	Mejoras y correcciones de errores Soluciona un problema por el que el administrador de secretos no aceptaba un ARN parcial.
2.1.7	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.4	Mejoras y correcciones de errores Corrige un problema en el que los secretos almacenados en caché se eliminaban cuando se implementaba el administrador de secretos y se reiniciaba el núcleo de Greengrass. Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.

Versión	Cambios
2.1.1	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Agrega compatibilidad con la integración de seguridad de hardware. El componente administrador de secretos puede cifrar y descifrar secretos mediante una clave privada que se almacena en un módulo de seguridad de hardware (HSM). Para obtener más información, consulte Integración de la seguridad de hardware. Mejoras y correcciones de errores • Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.9	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.5	Mejoras • Agregue soporte para las regiones y AWS GovCloud (US) regiones de AWS China.
2.0.4	Versión inicial.

Tunelización segura

Con el componente `aws.greengrass.SecureTunneling`, puede establecer una comunicación bidireccional segura con un dispositivo principal de Greengrass ubicado detrás de firewalls restringidos.

Por ejemplo, imagine que tiene un dispositivo principal de Greengrass detrás de un firewall que prohíbe todas las conexiones entrantes. La tunelización segura utiliza MQTT para transferir un token de acceso al dispositivo y, a continuación, se utiliza para establecer una conexión SSH con el dispositivo WebSockets a través del firewall. Con este túnel AWS IoT administrado, puede abrir la conexión SSH necesaria para su dispositivo. Para obtener más información sobre el uso de la tunelización AWS IoT segura para conectarse a dispositivos remotos, consulte la tunelización AWS IoT segura en la Guía para desarrolladores.AWS IoT

Este componente se suscribe al agente de mensajes AWS IoT Core MQTT sobre el `$aws/things/greengrass-core-device/tunnels/notify` tema para recibir notificaciones de tunelización segura.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Licencias
- De uso
- Véase también
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 1.1.x

- 1.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Arquitecturas:

- Armv71
- AArch64Armv8 ()
- x86_64

Requisitos

Este componente tiene los siguientes requisitos:

- Mínimo 32 MB de espacio en disco disponible para el componente de tunelización segura. Este requisito no incluye el software principal de Greengrass ni otros componentes que se ejecuten en el mismo dispositivo.
- Mínimo 16 MB de RAM disponible para el componente de tunelización segura. Este requisito no incluye el software principal de Greengrass ni otros componentes que se ejecuten en el mismo dispositivo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.
- Se requiere la versión 2.25 o posterior de la biblioteca C GNU (glibc) con un núcleo de Linux 3.2 o posterior para el componente de tunelización segura, versión 1.0.12 o posterior. No se admiten las versiones del sistema operativo y las bibliotecas que hayan superado su fecha de fin de vida útil. Debe utilizar un sistema operativo y bibliotecas compatibles a largo plazo.
- Tanto el sistema operativo como el motor de ejecución de Java deben estar instalados en 64 bits.
- Python 3.5 o posterior instalado en el dispositivo principal de Greengrass y agregado a la variable de entorno PATH.

- `libcrypto.so.1.1` instalado en el dispositivo principal de Greengrass y agregado a la variable de entorno PATH.
- Tráfico saliente abierto en el puerto 443 del dispositivo principal de Greengrass.
- Active la compatibilidad con el servicio de comunicación que desee utilizar para comunicarse con el dispositivo principal de Greengrass. Por ejemplo, para abrir una conexión SSH con el dispositivo, debe activar SSH en ese dispositivo.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`data.tunneling.iot.region.amazonaws.com`	443	Sí	Establezca túneles seguros.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

1.0.19 – 1.1.3

En la siguiente tabla se enumeran las dependencias de las versiones 1.0.19 a 1.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible

1.0.18

En la siguiente tabla, se muestran las dependencias de la versión 1.0.18 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible

1.0.16 – 1.0.17

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.16 a 1.0.17 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible

1.0.14 – 1.0.15

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.14 a 1.0.15 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible

1.0.11 – 1.0.13

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.11 a 1.0.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible

1.0.10

LEn la siguiente tabla, se muestran las dependencias de la versión 1.0.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible

1.0.9

En la siguiente tabla, se muestran las dependencias de la versión 1.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible

1.0.8

En la siguiente tabla, se muestran las dependencias de la versión 1.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible

1.0.5 - 1.0.7

En la siguiente tabla, se muestran las dependencias de las versiones 1.0.5 a 1.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible

1.0.4

En la siguiente tabla, se muestran las dependencias de la versión 1.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible

1.0.3

En la siguiente tabla, se muestran las dependencias de la versión 1.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible

1.0.2

En la siguiente tabla, se muestran las dependencias de la versión 1.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible

1.0.1

En la siguiente tabla, se muestran las dependencias de la versión 1.0.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

1.0.0

En la siguiente tabla, se muestran las dependencias de la versión 1.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`OS_DIST_INFO`

(Opcional) El sistema operativo del dispositivo principal. De forma predeterminada, el componente intenta identificar automáticamente el sistema operativo que se ejecuta en el dispositivo principal. Si el componente no se inicia con el valor predeterminado, utilice este valor para especificar el sistema operativo. Para ver una lista completa de los sistemas operativos admitidos, consulte Requisitos de los dispositivos.

Este valor puede ser uno de los siguientes: `auto`, `ubuntu`, `amzn2`, `raspberrypi`.

Valor predeterminado: `auto`

`accessControl`

(Opcional) El objeto que contiene la política de autorización que permite al componente suscribir mensajes en el tema de notificaciones de túnel seguro.

> **Note**
>
> No modifique este parámetro de configuración si la implementación se dirige a un grupo de objetos. Si su implementación se dirige a un dispositivo principal individual y desea restringir su suscripción al tema del dispositivo, especifique el nombre del objeto del dispositivo principal. En e valor `resources` de la política de autorización del dispositivo, sustituya el comodín del tema MQTT por el nombre del objeto del dispositivo.

```
{
```

```
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica que se debe permitir que este componente abra túneles seguros en un dispositivo principal denominado **`MyGreengrassCore`** que ejecuta Ubuntu.

```
{
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyala por la ruta a la carpeta raíz AWS IoT Greengrass .

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- AWS IoT Licencia de cliente de dispositivo/Apache 2.0
- AWS IoT Device SDK para Java/Apache License 2.0
- gson/Apache License 2.0
- log4j/Apache License 2.0
- slf4j/Apache License 2.0

De uso

Para usar el componente de tunelización segura en su dispositivo, haga lo siguiente:

1. Implemente el componente de tunelización segura en su dispositivo.
2. Abra la consola de AWS IoT. En el menú de la izquierda, seleccione Acciones remotas y, a continuación, seleccione Túneles seguros.
3. Cree un túnel para su dispositivo de Greengrass.
4. Descargue el token de acceso al origen.
5. Use el proxy local con el token de acceso al origen para conectarse a su destino. Para obtener más información, consulte Cómo usar el proxy local en la Guía del usuario de AWS IoT .

Véase también

- AWS IoT tunelización segura en la guía para desarrolladores AWS IoT
- Cómo usar el proxy local en la Guía para desarrolladores de AWS IoT

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.1.3	Mejoras y correcciones de errores • Actualiza el cliente de AWS IoT dispositivo subyacente invocado por el componente de la versión 1.10.0 a la versión 1.10.1. • También soluciona los problemas de compatibilidad con la bibliotec a GNU C (glibc) en la versión 1.1.2 del componente de tunelización segura.
1.1.2	**Warning** Esta versión ya no está disponible. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Esta versión actualiza el dispositivo de cliente de AWS IoT subyacente invocado por el componente de la versión 1.9.0 a la versión 1.10.0. • Soluciona el problema de transferencia de carga útil que impedía a los usuarios reenviar archivos de gran tamaño desde los dispositi vos principales de la versión 2 de Greengrass al dispositivo de origen mediante el túnel seguro.
1.1.1	Mejoras y correcciones de errores • Agrega una configuración compatible con la versión lite del núcleo de Greengrass.
1.1.0	Nuevas características • Agrega compatibilidad de recetas para la versión lite del núcleo de Greengrass

Versión	Cambios
1.0.19	Mejoras y correcciones de errores • Esta versión actualiza el dispositivo de cliente de AWS IoT subyacente invocado por el componente de la versión 1.8.0 a la versión 1.9.0. • Aumenta el límite de túneles simultáneos a 20 túneles a nivel de componente. • Aumenta el tiempo de espera predeterminado del AWS IoT Greengrass Core IPC de 3 a 10 segundos. ⚠ Warning Si utiliza el proxy local de tunelización segura como cliente de origen del túnel, no actualice el componente a esta versión hasta que también haya actualizado el proxy local a la versión 3.1.1 o posterior.
1.0.18	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
1.0.17	Mejoras y correcciones de errores • Corrige el problema de limpieza de subprocesos que impedía a los usuarios crearan túneles. Este componente ahora limpiará un hilo una vez que reciba la CloseTunnel señal o si el túnel ha caducado después de 12 horas.
1.0.16	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
1.0.15	Mejoras y correcciones de errores • Soluciona un problema de inicio para los usuarios que no tienen un directorio principal en el dispositivo. El componente de tunelización segura ahora se inicia sin crear un directorio para documentos de sombra.
1.0.14	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.

Versión	Cambios
1.0.13	Mejoras y correcciones de errores • Soluciona un problema por el que un proceso cliente huérfano impide que más de un túnel apunte al dispositivo.
1.0.12	Mejoras y correcciones de errores • Añade compatibilidad con x86_64 (AMD64) y ARMv8 (Aarch64) cuando se ejecuta en el sistema operativo Raspberry Pi.
1.0.11	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
1.0.10	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
1.0.9	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
1.0.8	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
1.0.7	Mejoras y correcciones de errores • Corrige un problema que provocaba que el componente se desconect ara al transferir archivos de gran tamaño a través de SCP.
1.0.6	Esta versión contiene correcciones de errores.
1.0.5	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
1.0.4	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
1.0.3	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
1.0.2	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.

Versión	Cambios
1.0.1	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
1.0.0	Versión inicial.

Administrador de sombras

El componente administrador de sombras (`aws.greengrass.ShadowManager`) habilita el servicio de sombra local en su dispositivo principal. El servicio de sombra local permite a los componentes utilizar la comunicación entre procesos para interactuar con las sombras locales. El componente de administrador de sombras gestiona el almacenamiento de los documentos ocultos locales y también gestiona la sincronización de los estados ocultos locales con el servicio AWS IoT Device Shadow.

Para obtener más información sobre cómo los dispositivos principales de Greengrass pueden interactuar con las sombras, consulte Interacción con las sombras de dispositivo.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.3.x
- 2.2.x
- 2.1.x

- 2.0.x

Tipo

Este componente es un componente de complemento (`aws.greengrass.plugin`). El núcleo de Greengrass ejecuta este componente en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de este componente en el dispositivo principal.

Este componente usa el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- (Opcional) Para sincronizar las sombras con el servicio AWS IoT Device Shadow, la AWS IoT política del dispositivo principal de Greengrass debe permitir las siguientes acciones de política AWS IoT Core clandestina:
  - `iot:GetThingShadow`
  - `iot:UpdateThingShadow`
  - `iot:DeleteThingShadow`

  Para obtener más información sobre estas AWS IoT Core políticas, consulte las acciones AWS IoT Core políticas en la Guía para AWS IoT desarrolladores.

  Para obtener más información sobre la AWS IoT política mínima, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales
- Se admite la ejecución del componente administrador de sombras en una VPC.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.3.13

En la siguiente tabla se enumeran las dependencias de la versión 2.3.13 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.17.0	Flexible

2.3.12

La siguiente tabla muestra las dependencias de la versión 2.3.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.17.0	Flexible

2.3.11

La siguiente tabla muestra las dependencias de la versión 2.3.11 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.16.0	Flexible

2.3.10

La siguiente tabla muestra las dependencias de la versión 2.3.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.15.0	Flexible

2.3.9

En la siguiente tabla, se muestran las dependencias de la versión 2.3.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.14.0	Flexible

2.3.5 – 2.3.8

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.5 a 2.3.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.13.0	Flexible

2.3.3 and 2.3.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.3 y 2.3.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.12.0	Flexible

2.3.2

En la siguiente tabla, se muestran las dependencias de la versión 2.3.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.11.0	Flexible

2.3.0 and 2.3.1

En la siguiente tabla, se muestran las dependencias de las versiones 2.3.0 y 2.3.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.5.0 <2.10.0	Flexible

2.2.3 and 2.2.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.2.3 y 2.2.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <3.0.0	Flexible

2.2.2

En la siguiente tabla, se muestran las dependencias de la versión 2.2.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.9.0	Flexible

2.2.1

En la siguiente tabla, se muestran las dependencias de la versión 2.2.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.8.0	Flexible

2.1.1 and 2.2.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.1 y 2.2.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.7.0	Flexible

2.0.5 - 2.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.5 a 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.6.0	Flexible

2.0.3 and 2.0.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.3 y 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.5.0	Flexible

2.0.1 and 2.0.2

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.1 y 2.0.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.4.0	Flexible

2.0.0

En la siguiente tabla, se muestran las dependencias de la versión 2.0.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.2.0 <2.3.0	Flexible

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.3.x

`strategy`

(Opcional) La estrategia que utiliza este componente para sincronizar las sombras entre y el dispositivo principal. AWS IoT Core

Este objeto contiene la siguiente información:

`type`

(Opcional) El tipo de estrategia que utiliza este componente para sincronizar las sombras entre AWS IoT Core y el dispositivo principal. Puede elegir entre las siguientes opciones:

- `realTime`— Sincronice las sombras AWS IoT Core cada vez que se produzca una actualización de sombras.
- `periodic`— Sincronice las sombras AWS IoT Core en un intervalo regular que especifique con el parámetro `delay` de configuración.

Valor predeterminado: `realTime`

`delay`

(Opcional) El intervalo en segundos en el que este componente sincroniza las sombras con AWS IoT Core, al especificar la estrategia de sincronización `periodic`.

> **Note**
>
> Este parámetro es necesario si no se especifica estrategia de sincronización `periodic`.

`synchronize`

(Opcional) Los ajustes de sincronización que determinan cómo se sincronizan las sombras con la Nube de AWS.

> **Note**
>
> Debe crear una actualización de configuración con esta propiedad para sincronizar las sombras con la Nube de AWS.

Este objeto contiene la siguiente información:

`coreThing`

(Opcional) Las sombras de dispositivo principal se van a sincronizar. Este objeto contiene la siguiente información:

`classic`

(Opcional) De forma predeterminada, el administrador de sombras sincroniza el estado local de la sombra clásica de su dispositivo principal con la Nube de AWS. Si no desea sincronizar la sombra de dispositivo clásica, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo principales con nombre que se van a sincronizar. Debe especificar los nombres exactos de las sombras.

> **Warning**
>
> El AWS IoT Greengrass servicio usa la sombra `AWSManagedGreengrassV2Deployment` denominada para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

`shadowDocumentsMap`

(Opcional) Las sombras de dispositivo adicionales que se van a sincronizar. El uso de este parámetro de configuración facilita la especificación de documentos de sombras. Le recomendamos que utilice este parámetro en lugar del objeto `shadowDocuments`.

> **Note**
>
> Si especifica un objeto `shadowDocumentsMap`, no debe especificar `shadowDocuments`.

Cada objeto contiene la siguiente información:

`thingName`

La configuración de sombra *`thingName`* para esta configuración de sombra.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

`namedShadows`

La lista de sombras con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`shadowDocuments`

(Opcional) La lista de sombras de dispositivo adicionales que se van a sincronizar. Le recomendamos que utilice el parámetro `shadowDocumentsMap` en su lugar.

> **Note**
>
> Si especifica un objeto `shadowDocuments`, no debe especificar `shadowDocumentsMap`.

Cada objeto en la lista contiene la siguiente información.

`thingName`

Nombre del objeto del dispositivo para el que se sincronizan las sombras.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`direction`

(Opcional) La dirección para sincronizar las sombras entre el servicio de sombra local y la Nube de AWS. Puede configurar esta opción para reducir el ancho de banda y las conexiones a la Nube de AWS. Puede elegir entre las siguientes opciones:

- `betweenDeviceAndCloud`: sincronice las sombras entre el servicio de sombra local y la Nube de AWS.
- `deviceToCloud`— Envía actualizaciones ocultas desde el servicio paralelo local al Nube de AWS e ignora las actualizaciones ocultas del Nube de AWS.
- `cloudToDevice`: recibe actualizaciones de sombras de la Nube de AWS y no envía actualizaciones de sombras del servicio de sombras local a la Nube de AWS.

Valor predeterminado: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Opcional) La configuración que determina los límites de tasa para las solicitudes de servicios de sombra.

Este objeto contiene la siguiente información:

`maxOutboundSyncUpdatesPerSecond`

(Opcional) El número máximo de solicitudes de sincronización por segundo que transmite el dispositivo.

Valor predeterminado: 100 solicitudes por segundo

`maxTotalLocalRequestsRate`

(Opcional) El número máximo de solicitudes de IPC locales por segundo que se envían al dispositivo principal.

Valor predeterminado: 200 solicitudes por segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) La cantidad máxima de solicitudes de IPC locales por segundo que se envían para cada objeto IoT conectado.

Predeterminado: 20 requests/second para cada cosa

> **Note**
>
> Estos parámetros de límites de tasa definen el número máximo de solicitudes por segundo para el servicio de sombra local. El número máximo de solicitudes por segundo para el servicio AWS IoT Device Shadow depende de usted Región de AWS. Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) El tamaño máximo permitido de cada documento de estado JSON para las sombras locales.

Si aumenta este valor debe incrementar también el límite de recursos del documento de estado JSON para sombras en la nube. Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

Valor predeterminado: 8192 bytes

Valor máximo: 30 720 bytes

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo, se muestra un ejemplo de actualización de la combinación de configuraciones con todos los parámetros de configuración disponibles para el componente administrador de sombras.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.2.x

`strategy`

(Opcional) La estrategia que utiliza este componente para sincronizar las sombras entre el dispositivo principal AWS IoT Core y el dispositivo principal.

Este objeto contiene la siguiente información:

`type`

(Opcional) El tipo de estrategia que utiliza este componente para sincronizar las sombras entre AWS IoT Core y el dispositivo principal. Puede elegir entre las siguientes opciones:

- `realTime`— Sincronice las sombras AWS IoT Core cada vez que se produzca una actualización de sombras.
- `periodic`— Sincronice las sombras AWS IoT Core en un intervalo regular que especifique con el parámetro `delay` de configuración.

Valor predeterminado: `realTime`

`delay`

(Opcional) El intervalo en segundos en el que este componente sincroniza las sombras con AWS IoT Core, al especificar la estrategia de sincronización `periodic`.

> **Note**
>
> Este parámetro es necesario si no se especifica estrategia de sincronización `periodic`.

`synchronize`

(Opcional) Los ajustes de sincronización que determinan cómo se sincronizan las sombras con la Nube de AWS.

> **Note**
>
> Debe crear una actualización de configuración con esta propiedad para sincronizar las sombras con la Nube de AWS.

Este objeto contiene la siguiente información:

`coreThing`

(Opcional) Las sombras de dispositivo principal se van a sincronizar. Este objeto contiene la siguiente información:

`classic`

(Opcional) De forma predeterminada, el administrador de sombras sincroniza el estado local de la sombra clásica de su dispositivo principal con la Nube de AWS. Si no desea sincronizar la sombra de dispositivo clásica, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo principales con nombre que se van a sincronizar. Debe especificar los nombres exactos de las sombras.

> **Warning**
>
> El AWS IoT Greengrass servicio usa la sombra `AWSManagedGreengrassV2Deployment` denominada para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

`shadowDocumentsMap`

(Opcional) Las sombras de dispositivo adicionales que se van a sincronizar. El uso de este parámetro de configuración facilita la especificación de documentos de sombras. Le recomendamos que utilice este parámetro en lugar del objeto `shadowDocuments`.

> **Note**
>
> Si especifica un objeto `shadowDocumentsMap`, no debe especificar `shadowDocuments`.

Cada objeto contiene la siguiente información:

`thingName`

La configuración de sombra *`thingName`* para esta configuración de sombra.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

`namedShadows`

La lista de sombras con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`shadowDocuments`

(Opcional) La lista de sombras de dispositivo adicionales que se van a sincronizar. Le recomendamos que utilice el parámetro `shadowDocumentsMap` en su lugar.

> **Note**
>
> Si especifica un objeto `shadowDocuments`, no debe especificar `shadowDocumentsMap`.

Cada objeto en la lista contiene la siguiente información.

`thingName`

Nombre del objeto del dispositivo para el que se sincronizan las sombras.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`direction`

(Opcional) La dirección para sincronizar las sombras entre el servicio de sombra local y la Nube de AWS. Puede configurar esta opción para reducir el ancho de banda y las conexiones a la Nube de AWS. Puede elegir entre las siguientes opciones:

- `betweenDeviceAndCloud`: sincronice las sombras entre el servicio de sombra local y la Nube de AWS.
- `deviceToCloud`— Envía actualizaciones ocultas desde el servicio paralelo local al Nube de AWS e ignora las actualizaciones ocultas del Nube de AWS.
- `cloudToDevice`: recibe actualizaciones de sombras de la Nube de AWS y no envía actualizaciones de sombras del servicio de sombras local a la Nube de AWS.

Valor predeterminado: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Opcional) La configuración que determina los límites de tasa para las solicitudes de servicios de sombra.

Este objeto contiene la siguiente información:

`maxOutboundSyncUpdatesPerSecond`

(Opcional) El número máximo de solicitudes de sincronización por segundo que transmite el dispositivo.

Valor predeterminado: 100 solicitudes por segundo

`maxTotalLocalRequestsRate`

(Opcional) El número máximo de solicitudes de IPC locales por segundo que se envían al dispositivo principal.

Valor predeterminado: 200 solicitudes por segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) La cantidad máxima de solicitudes de IPC locales por segundo que se envían para cada objeto IoT conectado.

Predeterminado: 20 requests/second para cada cosa

> **Note**
>
> Estos parámetros de límites de tasa definen el número máximo de solicitudes por segundo para el servicio de sombra local. El número máximo de solicitudes por segundo para el servicio AWS IoT Device Shadow depende de usted Región de AWS.

Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) El tamaño máximo permitido de cada documento de estado JSON para las sombras locales.

Si aumenta este valor debe incrementar también el límite de recursos del documento de estado JSON para sombras en la nube. Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

Valor predeterminado: 8192 bytes

Valor máximo: 30 720 bytes

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo, se muestra un ejemplo de actualización de la combinación de configuraciones con todos los parámetros de configuración disponibles para el componente administrador de sombras.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
```

```
            }
         },
         "direction":"betweenDeviceAndCloud"
      },
      "rateLimits":{
         "maxOutboundSyncUpdatesPerSecond":100,
         "maxTotalLocalRequestsRate":200,
         "maxLocalRequestsPerSecondPerThing":20
      },
      "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

`strategy`

(Opcional) La estrategia que utiliza este componente para sincronizar las sombras entre el dispositivo principal AWS IoT Core y el dispositivo principal.

Este objeto contiene la siguiente información:

`type`

(Opcional) El tipo de estrategia que utiliza este componente para sincronizar las sombras entre AWS IoT Core y el dispositivo principal. Puede elegir entre las siguientes opciones:

- `realTime`— Sincronice las sombras AWS IoT Core cada vez que se produzca una actualización de sombras.
- `periodic`— Sincronice las sombras AWS IoT Core en un intervalo regular que especifique con el parámetro `delay` de configuración.

Valor predeterminado: `realTime`

`delay`

(Opcional) El intervalo en segundos en el que este componente sincroniza las sombras con AWS IoT Core, al especificar la estrategia de sincronización `periodic`.

> **Note**
>
> Este parámetro es necesario si no se especifica estrategia de sincronización `periodic`.

`synchronize`

(Opcional) Los ajustes de sincronización que determinan cómo se sincronizan las sombras con la Nube de AWS.

> **Note**
>
> Debe crear una actualización de configuración con esta propiedad para sincronizar las sombras con la Nube de AWS.

Este objeto contiene la siguiente información:

`coreThing`

(Opcional) Las sombras de dispositivo principal se van a sincronizar. Este objeto contiene la siguiente información:

`classic`

(Opcional) De forma predeterminada, el administrador de sombras sincroniza el estado local de la sombra clásica de su dispositivo principal con la Nube de AWS. Si no desea sincronizar la sombra de dispositivo clásica, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo principales con nombre que se van a sincronizar. Debe especificar los nombres exactos de las sombras.

> **Warning**
>
> El AWS IoT Greengrass servicio usa la sombra `AWSManagedGreengrassV2Deployment` denominada para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

`shadowDocumentsMap`

(Opcional) Las sombras de dispositivo adicionales que se van a sincronizar. El uso de este parámetro de configuración facilita la especificación de documentos de sombras. Le recomendamos que utilice este parámetro en lugar del objeto `shadowDocuments`.

> **Note**
>
> Si especifica un objeto `shadowDocumentsMap`, no debe especificar `shadowDocuments`.

Cada objeto contiene la siguiente información:

`thingName`

La configuración de sombra *`thingName`* para esta configuración de sombra.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

`namedShadows`

La lista de sombras con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`shadowDocuments`

(Opcional) La lista de sombras de dispositivo adicionales que se van a sincronizar. Le recomendamos que utilice el parámetro `shadowDocumentsMap` en su lugar.

> **Note**
>
> Si especifica un objeto `shadowDocuments`, no debe especificar `shadowDocumentsMap`.

Cada objeto en la lista contiene la siguiente información.

`thingName`

Nombre del objeto del dispositivo para el que se sincronizan las sombras.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`rateLimits`

(Opcional) La configuración que determina los límites de tasa para las solicitudes de servicios de sombra.

Este objeto contiene la siguiente información:

`maxOutboundSyncUpdatesPerSecond`

(Opcional) El número máximo de solicitudes de sincronización por segundo que transmite el dispositivo.

Valor predeterminado: 100 solicitudes por segundo

`maxTotalLocalRequestsRate`

(Opcional) El número máximo de solicitudes de IPC locales por segundo que se envían al dispositivo principal.

Valor predeterminado: 200 solicitudes por segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) La cantidad máxima de solicitudes de IPC locales por segundo que se envían para cada objeto IoT conectado.

Predeterminado: 20 requests/second para cada cosa

> **Note**
>
> Estos parámetros de límites de tasa definen el número máximo de solicitudes por segundo para el servicio de sombra local. El número máximo de solicitudes por segundo para el servicio AWS IoT Device Shadow depende de usted Región de AWS.

Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) El tamaño máximo permitido de cada documento de estado JSON para las sombras locales.

Si aumenta este valor debe incrementar también el límite de recursos del documento de estado JSON para sombras en la nube. Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

Valor predeterminado: 8192 bytes

Valor máximo: 30 720 bytes

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo, se muestra un ejemplo de actualización de la combinación de configuraciones con todos los parámetros de configuración disponibles para el componente administrador de sombras.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
```

```
          }
       },
       "direction":"betweenDeviceAndCloud"
    },
    "rateLimits":{
       "maxOutboundSyncUpdatesPerSecond":100,
       "maxTotalLocalRequestsRate":200,
       "maxLocalRequestsPerSecondPerThing":20
    },
    "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

`synchronize`

(Opcional) Los ajustes de sincronización que determinan cómo se sincronizan las sombras con la Nube de AWS.

> **Note**
>
> Debe crear una actualización de configuración con esta propiedad para sincronizar las sombras con la Nube de AWS.

Este objeto contiene la siguiente información:

`coreThing`

(Opcional) Las sombras de dispositivo principal se van a sincronizar. Este objeto contiene la siguiente información:

`classic`

(Opcional) De forma predeterminada, el administrador de sombras sincroniza el estado local de la sombra clásica de su dispositivo principal con la Nube de AWS. Si no desea sincronizar la sombra de dispositivo clásica, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo principales con nombre que se van a sincronizar. Debe especificar los nombres exactos de las sombras.

> **Warning**
>
> El AWS IoT Greengrass servicio usa el `AWSManagedGreengrassV2Deployment` nombre shadow para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

`shadowDocumentsMap`

(Opcional) Las sombras de dispositivo adicionales que se van a sincronizar. El uso de este parámetro de configuración facilita la especificación de documentos de sombras. Le recomendamos que utilice este parámetro en lugar del objeto `shadowDocuments`.

> **Note**
>
> Si especifica un objeto `shadowDocumentsMap`, no debe especificar `shadowDocuments`.

Cada objeto contiene la siguiente información:

`thingName`

La configuración de sombra *`thingName`* para esta configuración de sombra.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

`namedShadows`

La lista de sombras con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`shadowDocuments`

(Opcional) La lista de sombras de dispositivo adicionales que se van a sincronizar. Le recomendamos que utilice el parámetro `shadowDocumentsMap` en su lugar.

> **Note**
>
> Si especifica un objeto `shadowDocuments`, no debe especificar `shadowDocumentsMap`.

Cada objeto en la lista contiene la siguiente información.

`thingName`

Nombre del objeto del dispositivo para el que se sincronizan las sombras.

`classic`

(Opcional) Si no desea sincronizar la sombra de dispositivo clásico con el dispositivo `thingName`, configúrela en `false`.

Valor predeterminado: `true`

`namedShadows`

(Opcional) La lista de sombras de dispositivo con nombre que desea sincronizar. Debe especificar los nombres exactos de las sombras.

`rateLimits`

(Opcional) La configuración que determina los límites de tasa para las solicitudes de servicios de sombra.

Este objeto contiene la siguiente información:

`maxOutboundSyncUpdatesPerSecond`

(Opcional) El número máximo de solicitudes de sincronización por segundo que transmite el dispositivo.

Valor predeterminado: 100 solicitudes por segundo

`maxTotalLocalRequestsRate`

(Opcional) El número máximo de solicitudes de IPC locales por segundo que se envían al dispositivo principal.

Valor predeterminado: 200 solicitudes por segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) La cantidad máxima de solicitudes de IPC locales por segundo que se envían para cada objeto IoT conectado.

Predeterminado: 20 requests/second para cada cosa

> **Note**
>
> Estos parámetros de límites de tasa definen el número máximo de solicitudes por segundo para el servicio de sombra local. El número máximo de solicitudes por segundo para el servicio AWS IoT Device Shadow depende de usted Región de AWS. Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) El tamaño máximo permitido de cada documento de estado JSON para las sombras locales.

Si aumenta este valor debe incrementar también el límite de recursos del documento de estado JSON para sombras en la nube. Para obtener más información, consulte los límites de la API del servicio de sombra de dispositivo AWS IoT en la Referencia general de Amazon Web Services.

Valor predeterminado: 8192 bytes

Valor máximo: 30 720 bytes

Example Ejemplo: actualización de la combinación de configuraciones

En el siguiente ejemplo, se muestra un ejemplo de actualización de la combinación de configuraciones con todos los parámetros de configuración disponibles para el componente administrador de sombras.

```
{
  "synchronize": {
    "coreThing": {
```

```
      "classic": true,
      "namedShadows": [
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": []
      }
    ]
  },
  "rateLimits": {
    "maxOutboundSyncUpdatesPerSecond": 100,
    "maxTotalLocalRequestsRate": 200,
    "maxLocalRequestsPerSecondPerThing": 20
  },
  "shadowDocumentSizeLimitBytes": 8192
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.3.13	Mejoras y correcciones de errores • Soluciona un problema que provocaba que las actualizaciones ocultas locales no se sincronizaran con la nube de forma intermitente, incluso cuando la configuración de la dirección de sincronización lo permitía. • Pequeñas correcciones de errores y mejoras.
2.3.12	Mejoras y correcciones de errores • Soluciona un problema por el que los despliegues se bloqueaban cuando se configuraban más de 1024 dispositivos ocultos.
2.3.11	Versión actualizada para el lanzamiento de la versión 2.15.0 de Greengrass nucleus.
2.3.10	Versión actualizada para la versión 2.14.0 de Greengrass Nucleus.
2.3.9	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.

Versión	Cambios
2.3.8	Mejoras y correcciones de errores • Soluciona un problema que provocaba que el administrador de sombras creara una situación de bloqueo durante la conexión del cliente MQTT.
2.3.7	Mejoras y correcciones de errores • Soluciona un problema por el que el administrador de sombras registrab a periódicamente un error `NullPointerException` durante la sincronización del administrador de sombras.
2.3.6	Mejoras y correcciones de errores • Corrige un problema por el que las propiedades de sombra que se eliminan mediante las actualizaciones de Nube de AWS mientras el dispositivo no está en línea siguen existiendo en la sombra local tras recuperar la conectividad.
2.3.5	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.3.4	Mejoras y correcciones de errores • Suma compatibilidad con documentos de estado de sombra nulos y vacíos.
2.3.3	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.3.2	Mejoras y correcciones de errores • Soluciona un problema por el que el administrador de sombras entra en estado `BROKEN` cuando la base de datos de sombra local está dañada. • Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.

Versión	Cambios
2.3.1	Mejoras y correcciones de errores • Soluciona una condición que podía impedir que las actualizaciones de sombra en la nube se sincronizaran. • Soluciona un problema por el que los cambios en la configuración de sincronización de sombras con nombre se aplicaban solo a una sombra con nombre.
2.3.0	Mejoras y correcciones de errores • Soluciona un problema que podía impedir que las sombras se sincroniz aran cuando la clave privada del dispositivo de Greengrass estaba almacenada en un módulo de seguridad de hardware.
2.2.4	Mejoras y correcciones de errores • Soluciona un problema por el que la validación del tamaño de la sombra no era coherente con el de la nube al actualizar el documento de sombra local. • Soluciona un problema por el que el administrador de sombras deja de escuchar las actualizaciones de la configuración si una implementación realiza un `RESET` en los nodos de configuración.
2.2.3	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.2.2	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.2.1	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.

Versión	Cambios
2.2.0	Nuevas características • Añade compatibilidad con el servicio oculto local a través de la publish/subscribe interfaz local. Ahora puede comunicarse con el intermediario de publish/subscribe mensajes local sobre temas ocultos de MQTT para obtener, actualizar y eliminar sombras en el dispositivo principal. Esta característica le permite conectar los dispositivos de cliente al servicio de sombra local mediante el puente MQTT para retransmitir mensajes sobre temas de sombra entre los dispositivos de cliente y la interfaz local de publicación/suscripción. Esta característica requiere la versión 2.6.0 o posterior del componente núcleo de Greengrass. Para conectar los dispositivos de cliente al servicio de sombra local, también debe utilizar la versión 2.2.0 o posterior del componente puente MQTT. • Agrega la opción `direction` que puede configurar para personalizar la dirección de sincronización de las sombras entre el servicio de sombra local y la Nube de AWS. Puede configurar esta opción para reducir el ancho de banda y las conexiones a la Nube de AWS.
2.1.1	Mejoras y correcciones de errores • Soluciona un problema por el que la profundidad máxima en las secciones `desired` y `reported` del documento de estado de sombra de dispositivo JSON era de 4 niveles en lugar de 5 niveles. • Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Agrega compatibilidad con intervalos periódicos de sincronización paralela, de modo que puede configurar el dispositivo principal para reducir el uso de ancho de banda y los cargos.
2.0.6	Esta versión contiene correcciones de errores y mejoras.
2.0.5	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.

Versión	Cambios
2.0.4	Mejoras y correcciones de errores • Soluciona un problema que provocaba que el administrador de sombras eliminara las versiones recién creadas de cualquier sombra que se hubiera eliminado anteriormente. • Actualiza la operación de IPC `DeleteThingShadow` para incrementar la versión de sombra cuando se invoca.
2.0.3	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.2	Mejoras y correcciones de errores • Se ha corregido un error que provocaba que el administrador de sombras no reconociera la propiedad `delta` al sincronizar los estados de sombra desde AWS IoT Core. • Se ha corregido un problema que a veces provocaba que las solicitudes de sincronización de una sombra se combinaran de forma incorrecta.
2.0.1	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.0	Versión inicial.

Amazon SNS

El componente Amazon SNS (`aws.greengrass.SNS`) publica mensajes en un tema de Amazon Simple Notification Service (Amazon SNS). Puede usar este componente para enviar eventos desde los dispositivos principales de Greengrass a servidores web, direcciones de correo electrónico y otros suscriptores de mensajes. Para obtener más información, consulte ¿Qué es Amazon SNS? en la Guía para desarrolladores de Amazon Simple Notification Service.

Para publicar en un tema de Amazon SNS con este componente, publique un mensaje en un tema al que se suscriba este componente. De forma predeterminada, este componente se suscribe al tema `sns/message` de publicación/suscripción local. Puede especificar otros temas, incluidos los de AWS IoT Core MQTT, al implementar este componente.

En su componente personalizado, puede que desee implementar una lógica de filtrado o formato para procesar los mensajes de otros orígenes antes de publicarlos en este componente. Esto le permite centralizar la lógica de procesamiento de mensajes en un único componente.

> **Note**
>
> Este componente proporciona una funcionalidad similar a la del conector Amazon SNS de la versión 1. AWS IoT Greengrass Para obtener información, consulte Amazon SNS connector en la Guía para desarrolladores de AWS IoT Greengrass V1.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada
- Datos de salida
- Archivo de registro local
- Licencias
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente de Lambda (`aws.greengrass.lambda`). El núcleo de Greengrass ejecuta la función de Lambda de este componente mediante el componente lanzador de Lambda.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Versión 3.7 de Python instalada en el dispositivo principal y agregada a la variable de entorno PATH.
- Un tema de Amazon SNS. Para obtener instrucciones, consulte el tema Creación de un tema de Amazon SNS en la Guía para desarrolladores de Amazon Simple Notification Service.
- El rol del dispositivo de Greengrass debe permitir la acción `sns:Publish`, tal como se muestra en la siguiente política de IAM de ejemplo.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012:topic-name"
      ]
```

```
    }
  ]
}
```

Puede anular dinámicamente el tema predeterminado en la carga útil del mensaje de entrada para este componente. Si su aplicación utiliza esta característica, la política de IAM debe incluir todos los temas de destino como recursos. Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *)

- Para recibir los datos de salida de este componente, debe combinar la siguiente actualización de configuración para el componente del enrutador de suscripción antiguo (`aws.greengrass.LegacySubscriptionRouter`) cuando implemente este componente. Esta configuración especifica el tema en el que este componente publica las respuestas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-sns": {
        "id": "aws-greengrass-sns",
        "source": "component:aws.greengrass.SNS",
        "subject": "sns/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-sns": {
        "id": "aws-greengrass-sns",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
        "subject": "sns/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  - *region*Sustitúyalo por el Región de AWS que utilice.

- *version*Sustitúyala por la versión de la función Lambda que ejecuta este componente. Para encontrar la versión de la función de Lambda, debe ver la receta de la versión de este componente que desee implementar. Abra la página de detalles de este componente en la consola de AWS IoT Greengrass y busque el par clave-valor de la función de Lambda. Este par clave-valor contiene el nombre y la versión de la función de Lambda.

> **Important**
>
> Debe actualizar la versión de la función de Lambda en el enrutador de suscripción antiguo cada vez que implemente este componente. Esto garantiza que utilice la versión correcta de la función de Lambda para la versión del componente que implemente.

Para obtener más información, consulte Crear implementaciones.

- Se admite la ejecución del componente Amazon SNS en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.
  - El componente Amazon SNS debe tener una conectividad con `sns.region.amazonaws.com` que tenga el punto de conexión de VPC de `com.amazonaws.us-east-1.sns`.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`sns.region.amazonaws.com`	443	Sí	Publicar mensajes en Amazon SNS.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.10

En la siguiente tabla, se muestran las dependencias de la versión 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.16.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.9

En la siguiente tabla, se muestran las dependencias de la versión 2.1.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 =2.0.0 <2.15.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.1.8

En la siguiente tabla, se muestran las dependencias de la versión 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.14.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.7

En la siguiente tabla, se muestran las dependencias de la versión 2.1.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.6

En la siguiente tabla, se muestran las dependencias de la versión 2.1.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.5

En la siguiente tabla, se muestran las dependencias de la versión 2.1.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.4

En la siguiente tabla, se muestran las dependencias de la versión 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.3

En la siguiente tabla, se muestran las dependencias de la versión 2.1.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.2

En la siguiente tabla, se muestran las dependencias de la versión 2.1.2 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.8 - 2.1.0

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.8 y 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	^2.0.0	Rígido

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.6

En la siguiente tabla, se muestran las dependencias de la versión 2.0.6 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.5

En la siguiente tabla, se muestran las dependencias de la versión 2.0.5 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.4

En la siguiente tabla, se muestran las dependencias de la versión 2.0.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Rígido
Lanzador de Lambda	^2.0.0	Rígido
Tiempos de ejecución de Lambda	^2.0.0	Flexible
Servicio de intercambio de token	^2.0.0	Rígido

2.0.3

En la siguiente tabla, se muestran las dependencias de la versión 2.0.3 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Rígido
Lanzador de Lambda	>=1.0.0	Rígido
Tiempos de ejecución de Lambda	>=1.0.0	Flexible
Servicio de intercambio de token	>=1.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

> **Note**
>
> La configuración predeterminada de este componente incluye los parámetros de la función de Lambda. Le recomendamos que edite solo los siguientes parámetros para configurar este componente en sus dispositivos.

`lambdaParams`

Un objeto que contiene los parámetros de la función de Lambda de este componente. Este objeto contiene la siguiente información:

`EnvironmentVariables`

Un objeto que contiene los parámetros de la función de Lambda. Este objeto contiene la siguiente información:

`DEFAULT_SNS_ARN`

El ARN del tema predeterminado de Amazon SNS en el que este componente publica los mensajes. Puede anular el tema de destino con la propiedad `sns_topic_arn` en la carga útil de mensajes de entrada.

`containerMode`

(Opcional) El modo de almacenamiento en contenedores de este componente. Puede elegir entre las siguientes opciones:

- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.
- `GreengrassContainer`— El componente se ejecuta en un entorno de ejecución aislado dentro del contenedor. AWS IoT Greengrass

Valor predeterminado: `GreengrassContainer`

`containerParams`

(Opcional) Un objeto que contiene los parámetros de contenedor de este componente. El componente utiliza estos parámetros si se especifica `GreengrassContainer` para `containerMode`.

Este objeto contiene la siguiente información:

`memorySize`

(Opcional) La cantidad de memoria (en kilobytes) que se va a asignar al componente.

Predeterminado de 512 MB (525 312 KB).

`pubsubTopics`

(Opcional) Un objeto que contiene los temas a los que el componente se suscribe para recibir mensajes. Puede especificar cada tema y si el componente se suscribe a temas de MQTT AWS IoT Core o a temas locales publish/subscribe .

Este objeto contiene la siguiente información:

`0`: se trata de un índice de matriz en forma de cadena.

Un objeto que contiene la siguiente información:

`type`

(Opcional) El tipo de publish/subscribe mensajes que utiliza este componente para suscribirse a los mensajes. Puede elegir entre las siguientes opciones:

- `PUB_SUB` — Suscribirse a la mensajería de publicación/suscripción local. Si elige esta opción, el tema no podrá contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde un componente personalizado cuando especifique esta opción, consulte Publicar/suscribir mensajes locales.
- `IOT_CORE`— Suscríbase a los mensajes de AWS IoT Core MQTT. Si elige esta opción, el tema puede contener caracteres comodín de MQTT. Para obtener más información sobre cómo enviar mensajes desde componentes personalizados cuando especifique esta opción, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Valor predeterminado: `PUB_SUB`

`topic`

(Opcional) El tema al que se suscribe el componente para recibir mensajes. Si especifica `IotCore` para `type`, puede usar los comodines de MQTT (+ y #) en este tema.

Example Ejemplo: actualización de la combinación de configuraciones (modo en contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Ejemplo: actualización de la combinación de configuraciones (modo sin contenedor)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

Datos de entrada

Este componente acepta mensajes sobre el siguiente tema y publica el mensaje tal cual en el tema de Amazon SNS de destino. De forma predeterminada, este componente se suscribe a los mensajes locales publish/subscribe . Para obtener más información sobre cómo publicar mensajes en este componente desde sus componentes personalizados, consulte Publicar/suscribir mensajes locales.

Tema predeterminado (publicación/suscripción local): `sns/message`

El mensaje acepta las siguientes propiedades. Los mensajes de entrada deben tener un formato JSON válido.

`request`

Información sobre los mensajes que se envían al tema de Amazon SNS.

Tipo: `object` que contiene la siguiente información:

`message`

El contenido del mensaje como una cadena.

Para enviar un objeto JSON, serialícelo como una cadena y especifique `json` para la propiedad `message_structure`.

Tipo: `string`

`subject`

(Opcional) El asunto del mensaje.

Tipo: `string`

El asunto puede ser texto ASCII y tener hasta 100 caracteres. Debe empezar por una letra, un número o un signo de puntuación. No debe incluir saltos de línea ni caracteres de control.

`sns_topic_arn`

(Opcional) El ARN del tema de Amazon SNS en el que este componente publica los mensajes. Especifique esta propiedad para anular el tema predeterminado de Amazon SNS.

Tipo: `string`

`message_structure`

(Opcional) La estructura del mensaje. Especifique `json` que se envíe un mensaje JSON que se serialice como una cadena en la propiedad `content`.

Tipo: `string`

Valores válidos: `json`

`id`

Un ID arbitrario para la solicitud. Use esta propiedad para asignar una solicitud de entrada a una respuesta de salida. Si especifica esta propiedad, el componente establece la propiedad `id` en el objeto de respuesta para este valor.

Tipo: `string`

> **Note**
>
> El tamaño del mensaje puede ser de 256 KB como máximo.

Example Ejemplo de entrada: mensaje en cadena

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example Ejemplo de entrada: mensaje JSON

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
```

```
  },
  "id": "request123"
}
```

Datos de salida

Este componente publica las respuestas como datos de salida sobre el siguiente tema MQTT de forma predeterminada. Debe especificar este tema como parte de `subject` en la configuración del componente antiguo del enrutador de suscripciones. Para obtener más información sobre cómo suscribirse a los mensajes sobre este tema en sus componentes personalizados, consulte Publicar/suscribir mensajes MQTT AWS IoT Core.

Tema predeterminado (AWS IoT Core MQTT): `sns/message/status`

Example Ejemplo de salida: Correcto

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example Ejemplo de salida: Error

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyalo por la ruta a la carpeta AWS IoT Greengrass raíz.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
```

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- AWS SDK para Python (Boto3)/Apache License 2.0
- botocore/Apache License 2.0
- dateutil/PSF License
- docutils/BSD License, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- jmespath/MIT License
- s3transfer/Apache License 2.0
- urllib3/MIT License

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.10	Versión actualizada para la versión 2.15.0 de Greengrass nucleus.
2.1.9	Versión actualizada para la versión 2.14.0 de Greengrass nucleus.
2.1.8	Versión actualizada para el lanzamiento de la versión 2.13.0 del núcleo de Greengrass.

Versión	Cambios
2.1.7	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.6	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.
2.1.4	Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.3	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.2	Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Suma compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red y Permita que el dispositivo principal confíe en un proxy HTTPS.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.7	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.6	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.

Versión	Cambios
2.0.5	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.4	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.3	Versión inicial.

Administrador de flujos

El componente administrador de flujos (`aws.greengrass.StreamManager`) le permite procesar flujos de datos para transferirlos Nube de AWS desde los dispositivos principales de Greengrass.

Para obtener más información acerca de cómo configurar y usar el administrador de flujos en componentes personalizados, consulte Administración de flujos de datos en los dispositivos principales de Greengrass.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.2.x
- 2.1.x
- 2.0.x

> **Note**
>
> Si usa el administrador de flujos para exportar datos a la nube, no puede actualizar la versión 2.0.7 del componente de administrador de flujos a una versión entre la 2.0.8 y la 2.0.11. Si implementa el administrador de flujos por primera vez, le recomendamos que implemente la última versión del componente administrador de flujos.

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- La función de intercambio de fichas debe permitir el acceso a los Nube de AWS destinos que utilice con el administrador de transmisiones. Para obtener más información, consulte lo siguiente:
  - the section called “AWS IoT Analytics canales”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “AWS IoT SiteWise propiedades de los activos”
  - the section called “Objetos de Amazon S3”
- Se admite la ejecución del componente administrador de flujos en una VPC. Para implementar este componente en una VPC, se requiere lo siguiente.
  - El componente del administrador de transmisiones debe tener conectividad con el AWS servicio en el que publicas los datos.

  - Amazon S3: `com.amazonaws.region.s3`
  - Amazon Kinesis Data Streams: `com.amazonaws.region.kinesis-streams`
  - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
- Si publica datos en Amazon S3 en la región `us-east-1`, este componente intentará utilizar el punto de conexión global de S3 de forma predeterminada; sin embargo, este punto de conexión no está disponible a través del punto de conexión de la interfaz de VPC de Amazon S3. Para obtener más información, consulte Restricciones y limitaciones AWS PrivateLink de Amazon S3. Para resolver este problema, puede elegir entre las siguientes opciones.
  - Configure el componente de administrador de flujos para que utilice el punto de conexión S3 de la región `us-east-1`, configurando `-Daws.s3UseUsEast1RegionalEndpoint=regional` en `JVM_ARGS`.
  - Cree un punto de conexión de VPC de puerta de enlace de Amazon S3 en lugar de un punto de conexión de VPC de interfaz de Amazon S3. Los puntos de conexión de la puerta de enlace S3 permiten el acceso al punto de conexión global de S3. Para obtener más información, consulte Creación de un punto de conexión de un gateway.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`iotanalytics.region.amazonaws.com`	443	No	Obligatorio si publicas datos en AWS IoT Analytics.
`kinesis.region.amazonaws.com`	443	No	Obligatorio si publica

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			datos en Firehose.
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	No	Obligatorio si publicas datos en AWS IoT SiteWise.
`*.s3.amazonaws.com`	443	No	Obligatorio si publica datos en buckets de S3. Puede sustituirlos por * con el nombre de cada bucket en el que publique los datos.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia.

También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.3 – 2.2.1

En la siguiente tabla se enumeran las dependencias de las versiones 2.1.3 a 2.2.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <3.0.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.11 – 2.1.12

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.11 a 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.13.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.9 – 2.1.10

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.9 a 2.1.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.12.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.5 – 2.1.8

En l a siguiente tabla, se muestran las dependencias de las versiones 2.1.5 a 2.1.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.11.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.2 – 2.1.4

En la siguiente tabla, se muestran las dependencias de las versiones 2.1.2 a 2.1.4 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.10.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.1

En la siguiente tabla, se muestran las dependencias de la versión 2.1.1 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.9.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.1.0

En la siguiente tabla, se muestran las dependencias de la versión 2.1.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.8.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.15

En la siguiente tabla, se muestran las dependencias de la versión 2.0.15 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.7.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.13 and 2.0.14

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.13 y 2.0.14 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.6.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.11 and 2.0.12

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.11 y 2.0.12 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.5.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.10

En la siguiente tabla, se muestran las dependencias de la versión 2.0.10 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.4.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.9

En la siguiente tabla, se muestran las dependencias de la versión 2.0.9 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.3.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.8

En la siguiente tabla, se muestran las dependencias de la versión 2.0.8 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.0 <2.2.0	Flexible

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=0.0.0	Rígido

2.0.7

En la siguiente tabla, se muestran las dependencias de la versión 2.0.7 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.0.3 <2.1.0	Flexible
Servicio de intercambio de token	>=0.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`STREAM_MANAGER_STORE_ROOT_DIR`

(Opcional) La ruta absoluta del directorio local utilizado para almacenar flujos. Este valor debe comenzar con una barra inclinada (por ejemplo, `/data`).

Debe especificar una carpeta existente y el usuario del sistema que ejecuta el componente de administrador de flujos debe tener permisos para leer y escribir en esta carpeta. Por ejemplo, puede ejecutar los siguientes comandos para crear y configurar una carpeta, `/var/greengrass/streams`, que especifique como carpeta raíz del administrador de flujos. Estos comandos permiten al usuario predeterminado del sistema, `ggc_user`, leer y escribir en esta carpeta.

```
sudo mkdir /var/greengrass/streams
```

```
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Valor predeterminado: */greengrass/v2*/work/aws.greengrass.StreamManager

`STREAM_MANAGER_SERVER_PORT`

(Opcional) El número de puerto local utilizado para comunicarse con el administrador de flujos.

Puede especificar `0` para usar un puerto disponible asignado al azar.

Valor predeterminado: `8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(Opcional) Puede hacer que sea obligatorio que los clientes se autentiquen antes de poder interactuar con administrador de flujos. El SDK del administrador de flujos controla la interacción entre los clientes y el administrador de flujos. Este parámetro determina qué clientes pueden llamar al SDK del administrador de flujos para trabajar con flujos. Para obtener más información, consulte Autenticación del cliente con el administrador de flujos.

Si especifica `true`, el SDK del administrador de flujos solo permite como clientes los componentes de Greengrass.

Si especifica `false`, el SDK del administrador de flujos permite que todos los procesos del dispositivo principal sean clientes.

Valor predeterminado: `true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(Opcional) El ancho de banda máximo promedio (en kilobits por segundo) que el administrador de flujos puede utilizar para exportar datos.

Valor predeterminado: sin límite

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(Opcional) Cantidad máxima de subprocesos activos que el administrador de flujos puede utilizar para exportar datos.

El tamaño óptimo depende del hardware, el volumen de secuencias y la cantidad planificada de secuencias de exportación. Si la velocidad de exportación es lenta, puede ajustar esta

configuración para encontrar el tamaño óptimo para su hardware y su caso de negocio. La CPU y la memoria del hardware del dispositivo principal son factores limitantes. Para comenzar, puede intentar establecer este valor igual a la cantidad de núcleos de procesador en el dispositivo.

Tenga cuidado de no establecer un tamaño superior al que admite el hardware. Cada flujo utiliza recursos de hardware, por lo que debe intentar limitar la cantidad de flujos de exportación en dispositivos restringidos.

Predeterminado: 5 subprocesos

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

(Opcional) El tamaño mínimo (en bytes) de una parte en una carga multiparte a Amazon S3. El administrador de flujos utiliza esta configuración y el tamaño del archivo de entrada para determinar cómo agrupar los datos en una solicitud PUT de varias partes.

> **Note**
>
> El administrador de flujos usa la propiedad de flujos `sizeThresholdForMultipartUploadBytes` para determinar si se debe exportar a Amazon S3 como una carga única o multiparte. Los componentes de AWS IoT Greengrass pueden establecer este umbral cuando crean un flujo que se exporta a Amazon S3.

Valor predeterminado: `5242880` (5 MB). Es el valor mínimo permitido.

`LOG_LEVEL`

(Opcional) El nivel de registro del componente. Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

`JVM_ARGS`

(Opcional) Argumentos personalizados de la máquina virtual de Java para pasar al administrador de flujos al inicio. Separe varios argumentos por espacios.

Utilice este parámetro sólo cuando deba anular la configuración predeterminada utilizada por la JVM. Por ejemplo, puede que necesite aumentar el tamaño predeterminado del montón si planea exportar un gran número de secuencias.

`startupTimeoutSeconds`

(Opcional) El tiempo máximo en segundos para que se inicie el componente. El estado del componente cambia a `ERRORED` si supera este tiempo de espera.

Valor predeterminado: `120`

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica el uso de un puerto no predeterminado.

```
{
  "STREAM_MANAGER_SERVER_PORT": "18088"
}
```

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la AWS IoT Greengrass carpeta raíz.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 -
Wait
```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.2.1	Mejoras y correcciones de errores • Soluciona un problema que impedía que el administrador de flujos exportara los mensajes a los destinos del flujo de datos de Kinesis. • Mejoras adicionales en el rendimiento de las exportaciones del administrador de flujos a los destinos del flujo de datos de Kinesis.
2.2.0	Nuevas características • Agrega una nueva clave de configuración para el tiempo de espera de inicio. El valor predeterminado es 120 segundos. • Agrega compatibilidad de recetas para la versión lite del núcleo de Greengrass
2.1.13	Mejoras y correcciones de errores Soporta puntos finales FIPS para AWS IoT SiteWise
2.1.12	Mejoras y correcciones de errores Actualiza el orden en que se utilizan las credenciales, de modo que las credenciales de Greengrass son las preferidas para las solicitudes de AWS servicio.

Versión	Cambios
2.1.11	Versión actualizada para el lanzamiento de la versión 2.12.0 del núcleo de Greengrass.
2.1.10	Mejoras y correcciones de errores Corrige un problema por el que la configuración del proxy HTTPS no confiaba en la cadena de certificados de la autoridad de certificación (CA) de Greengrass.
2.1.9	Versión actualizada para el lanzamiento de la versión 2.11.0 del núcleo de Greengrass.
2.1.8	Mejoras y correcciones de errores Soluciona un problema que provocaba que el administrador de transmisiones volviera a intentar SiteWise exportar de forma infinita y fallaba. `InvalidRequestException`
2.1.7	Mejoras y correcciones de errores Soluciona un problema por el que el administrador de flujos no podía leer correctamente la configuración del proxy.
2.1.6	Mejoras y correcciones de errores Soluciona un problema que podía provocar un bloqueo al arrancar algunos ARMv8 procesadores, incluido el Jetson Nano.
2.1.5	Versión actualizada para el lanzamiento de la versión 2.10.0 del núcleo de Greengrass.

Versión	Cambios
2.1.4	Mejoras y correcciones de errores • Soluciona un problema que provocaba que las entradas del mismo activo inmobiliario con la misma marca de tiempo dentro de un mismo lote se devolvieran `ConflictingOperationException` desde la SiteWise API, lo que provocaba que el administrador de transmisiones tuviera que volver a intentarlo continuamente. • Actualiza el tiempo de espera de conexión predeterminado de 3 segundos a 1 minuto.
2.1.3	Mejoras y correcciones de errores Corrige un problema de inicio en el sistema operativo Windows cuando se ejecuta como usuario SYSTEM.
2.1.2	Mejoras y correcciones de errores • Corrige un problema en el que el sistema operativo Windows utilizaba un idioma distinto del inglés. • Versión actualizada para el lanzamiento de la versión 2.9.0 del núcleo de Greengrass.
2.1.1	Versión actualizada para el lanzamiento de la versión 2.8.0 del núcleo de Greengrass.
2.1.0	Nuevas características • Actualiza este componente para enviar automáticamente las métricas de telemetría a Amazon. EventBridge Para obtener más información, consulte Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass. Esta característica requiere la versión 2.7.0 o posterior del componente núcleo de Greengrass. • Versión actualizada para el lanzamiento de la versión 2.7.0 del núcleo de Greengrass.

Versión	Cambios
2.0.15	Versión actualizada para el lanzamiento de la versión 2.6.0 del núcleo de Greengrass.
2.0.14	Esta versión contiene correcciones de errores y mejoras.
2.0.13	Versión actualizada para el lanzamiento de la versión 2.5.0 del núcleo de Greengrass.
2.0.12	Mejoras y correcciones de errores Corrige un problema que impedía actualizar el administrador de flujos versión 2.0.7 a una versión entre la versión 2.0.8 y la versión 2.0.11. Si utiliza el administrador de flujos para exportar datos a la nube, ahora puede actualizar a la versión 2.0.12.
2.0.11	Versión actualizada para el lanzamiento de la versión 2.4.0 del núcleo de Greengrass.
2.0.10	Versión actualizada para el lanzamiento de la versión 2.3.0 del núcleo de Greengrass.
2.0.9	Versión actualizada para el lanzamiento de la versión 2.2.0 del núcleo de Greengrass.
2.0.8	Versión actualizada para el lanzamiento de la versión 2.1.0 del núcleo de Greengrass.
2.0.7	Versión inicial.

Reenviador de registros del sistema

El reenviador de registros del sistema (`aws.greengrass.SystemLogForwarder`) carga los registros activos del sistema directamente a Amazon CloudWatch mediante la API CloudWatch HTTPS.

Important

Este componente solo reenviará los registros de systemd-journald generados durante el tiempo de ejecución. Para obtener más información sobre los registros de systemd-journald, consulte systemd-journald y journalctl.

Note

Este componente requiere permisos específicos para crear y administrar grupos de CloudWatch registros y transmisiones.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Puntos de conexión y puertos
- Dependencias
- Configuración
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente debe instalarse en sistemas Linux basados en systemd.

Requisitos

Este componente tiene los siguientes requisitos:

El componente requiere acceso para crear grupos de registros y transmisiones CloudWatch , así como permiso para realizar la llamada PutLogEvents HTTP. Debe agregar, como mínimo, los siguientes permisos de política al alias de rol de su dispositivo de Greengrass:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogGroup"],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/
systemLogs:*"
    },
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogStream", "logs:PutLogEvents"],
      "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:greengrass/
systemLogs:log-stream:${credentials-iot:ThingName}"
    }
  ]
}
```

> **Note**
>
> Para obtener más información, consulte la página de Github del reenviador de registros del sistema.

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`logs.region.amazonaws.com`	443	No	Es obligatorio si escribes CloudWatch registros en Logs.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

2.1.x

La siguiente tabla muestra las dependencias de la versión 2.1.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	Mayor or igual a 2.0.0	Rígido

Dependencia	Versiones compatibles	Tipo de dependencia
Versión lite del núcleo de Greengrass	>=2.3.0	Flexible

2.0.x

En la siguiente tabla, se muestran las dependencias de la versión 2.0.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	Mayor or igual a 2.0.0	Rígido

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

2.0.x-2.1.x

`maxUploadIntervalSec`

El periodo máximo durante el cual el reenviador de registros del sistema intentará cargar los registros. Dado que el reenviador de registros cargará los registros cuando la memoria se llene, es posible que siga cargando con más frecuencia que la cadencia máxima configurada.

`maxRetriesCount`

Cantidad de veces que el reenviador de registros del sistema intentará volver a intentar un error HTTP transitorio.

`bufferCapacity`

Tamaño del búfer circular para el almacenamiento de registros en memoria.

`logGroup`

La ruta de inicio de sesión. CloudWatch

`logStream`

El CloudWatch LogStream.

`filters`

Mapa de configuraciones de filtros para el dispositivo principal.

`services`

Lista de filtros de nombres de servicio que el reenviador de registros del sistema utilizará para determinar los registros que se cargarán. Un registro solo se cargará si el servicio desde el que se originó coincide con al menos uno de los filtros de esta lista. Los filtros de esta lista pueden ser una cadena con la que el nombre del servicio debe coincidir, o puede ser una cadena que termina en *, lo que significa que el prefijo debe coincidir.

Valor predeterminado: `[ggl.*]`

> **Important**
>
> Un registro solo se cargará si el servicio desde el que se originó coincide con al menos uno de los filtros de esta lista.

> **Note**
>
> Si se utiliza el valor *, se incluirán todos los servicios disponibles.

Example Ejemplo de configuración:

El siguiente ejemplo filtrará los registros por todos los servicios incluidos en la versión lite del núcleo de Greengrass.

```
{
    "maxUploadIntervalSec": 300,
    "maxRetriesCount": 3,
    "bufferCapacity": 1048576,
    "logGroup": "greengrass/systemLogs",
    "logStream": "deviceName",
    "filters": {
```

```
        "services": ["ggl.*"]
    }
}
```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.1.0	Mejoras y correcciones de errores • Actualiza la receta del componente para que sea compatible correctam ente con el núcleo de Greengrass. • Se ha mejorado el resultado del registro cuando no hay registros que cargar. • Corrección de errores y mejoras generales.
2.0.1	Mejoras y correcciones de errores Actualiza la receta del componente para que sea compatible correctam ente con los sistemas aarch64 (arm64).
2.0.0	Versión inicial.

Agente de Systems Manager

El componente AWS Systems Manager Agent (`aws.greengrass.SystemsManagerAgent`) instala el agente de Systems Manager para que pueda administrar los dispositivos principales con Systems Manager. Systems Manager es un AWS servicio que puede utilizar para ver y controlar su infraestructura AWS, incluidas las EC2 instancias de Amazon, los servidores y máquinas virtuales locales (VMs) y los dispositivos periféricos. Systems Manager le permite ver los datos operativos, automatizar las tareas operativas y mantener la seguridad y la conformidad. Para obtener más información, consulte ¿Qué es? AWS Systems Manager y Acerca de Systems Manager Agent en la Guía AWS Systems Manager del usuario.

Las herramientas y características de Systems Manager se denominan capacidades. Los dispositivos principales de Greengrass admiten todas las capacidades de Systems Manager. Para obtener más

información sobre estas capacidades y sobre cómo usar Systems Manager para administrar los dispositivos principales, consulte Capacidades de Systems Manager en la Guía del usuario de AWS Systems Manager .

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Véase también
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente solo se puede instalar en los dispositivos principales de Linux.

Requisitos

Este componente tiene los siguientes requisitos:

- Un dispositivo principal de Greengrass que se ejecuta en una plataforma Linux de 64 bits: Armv8 () AArch64 o x86_64.
- Debe tener una función de servicio AWS Identity and Access Management (IAM) que pueda asumir Systems Manager. Esta función debe incluir la política SSMManaged InstanceCore gestionada por Amazon o una política personalizada que defina permisos equivalentes. Para obtener más información, consulte Creación de un rol de servicio de IAM para dispositivos periféricos en la Guía del usuario de AWS Systems Manager .

  Al implementar este componente, debe especificar el nombre de este rol para el parámetro de configuración `SSMRegistrationRole`.
- El rol de dispositivo de Greengrass debe permitir las acciones `ssm:AddTagsToResource` y`ssm:RegisterManagedInstance`. El rol de dispositivo también debe permitir la acción `iam:PassRole` del rol de servicio de IAM que cumpla con el requisito anterior. El siguiente ejemplo de política de IAM concede estos permisos.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
```

```
}
```

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`ec2messages.` *`region`*`.amazonaws.com`	443	Sí	Comuníquese con el servicio de Systems Manager en la Nube de AWS.
`ssm.`*`region`*`.amazonaws.com`	443	Sí	Registre el dispositivo principal como un nodo administrado por Systems Manager.
`ssmmessages.` *`region`*`.amazonaws.com`	443	Sí	Comuníquese con Session Manager, una capacidad

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			de Systems Manager, en la Nube de AWS.

Para obtener más información, consulte Referencia: ec2messages, ssmmessages y otras llamadas a la API en la Guía del usuario de AWS Systems Manager .

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

En la siguiente tabla se enumeran las dependencias de las versiones 1.0.0 a 1.3.0 de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=2.0.0 <3.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`SSMRegistrationRole`

La función de servicio de IAM que puede asumir Systems Manager y que incluye la política SSMManaged InstanceCore gestionada por Amazon o una política personalizada que defina permisos equivalentes. Para obtener más información, consulte Creación de un rol de servicio de IAM para dispositivos periféricos en la Guía del usuario de AWS Systems Manager .

`SSMOverrideExistingRegistration`

(Opcional) Si el dispositivo principal ya ejecuta el Agente de Systems Manager registrado con una activación híbrida, puede anular el registro del Agente de Systems Manager existente del dispositivo. Defina esta opción en `true` para registrar el dispositivo principal como nodo administrado mediante el Agente de Systems Manager que proporciona este componente.

> **Note**
>
> Esta opción solo se aplica a los dispositivos que están registrados con una activación híbrida. Si el dispositivo principal se ejecuta en una EC2 instancia de Amazon con el Agente de Systems Manager instalado y un rol de perfil de instancia configurado, el ID de nodo gestionado existente de la EC2 instancia de Amazon empieza por`i-`. Al instalar el componente Agente de Systems Manager, el agente de Systems Manager registra un nuevo nodo administrado cuyo identificador empieza por `mi-` en lugar de `i-`. A continuación, puede utilizar el nodo administrado cuyo ID comienza con `mi-` para administrar el dispositivo principal con Systems Manager.

Valor predeterminado: `false`

`SSMResourceTags`

(Opcional) Las etiquetas que se van a agregar al nodo administrado por Systems Manager que este componente crea para el dispositivo principal. Puede usar estas etiquetas para administrar grupos de dispositivos principales con Systems Manager. Por ejemplo, puede ejecutar un comando en todos los dispositivos que tengan una etiqueta que especifique.

Especifique una lista en la que cada etiqueta sea un objeto con una `Key` y un `Value`. Por ejemplo, el siguiente valor de `SSMResourceTags` indica a este componente que establezca la etiqueta **`Owner`** en **`richard-roe`** en el nodo administrado del dispositivo principal.

```
[
  {
```

```
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

Este componente ignora estas etiquetas si el nodo administrado ya existe y `SSMOverrideExistingRegistration` es `false`.

Example Ejemplo: actualización de la combinación de configuraciones

El siguiente ejemplo de configuración especifica el uso de un rol de servicio denominado `SSMServiceRole` para permitir que el dispositivo principal se registre y se comunique con Systems Manager.

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

Archivo de registro local

El software del Agente de Systems Manager escribe los registros en una carpeta fuera de la carpeta raíz de Greengrass. Para obtener más información, consulte Visualización de registros de Systems Manager Agent en la Guía del usuario de AWS Systems Manager .

El componente Agente de Systems Manager utiliza scripts de intérprete de comandos para instalar, iniciar y detener el agente de Systems Manager. Puede encontrar el resultado de estos scripts en el siguiente archivo de registro.

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. */greengrass/v2*Sustitúyalo por la ruta a la carpeta AWS IoT Greengrass raíz.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

Véase también

- Administración de los dispositivos principales de Greengrass con AWS Systems Manager
- ¿Qué es AWS Systems Manager? en la Guía del usuario de AWS Systems Manager
- Información del Agente de Systems Manager en la Guía del usuario de AWS Systems Manager

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.3.0	Mejoras y correcciones de errores Actualiza este componente para obtener la versión 3.3.1142.0 del agente.
1.2.4	Mejoras y correcciones de errores Actualiza este componente para obtener la versión 3.2.2303.0 del Agente.
1.2.3	Mejoras y correcciones de errores • Agrega reintentos para la instalación del componente Agente con Snap en Greengrass. • Actualiza la configuración del componente Agente para usar solo la identidad Onprem en Greengrass. • Actualiza este componente para actualizar el agente solo cuando la versión del agente instalada no coincide con la versión del componente Agente SSM de Greengrass.

Versión	Cambios
1.1.0	Esta versión contiene correcciones de errores y mejoras.
1.0.0	Versión inicial.

Servicio de intercambio de token

El componente del servicio de intercambio de tokens (`aws.greengrass.TokenExchangeService`) proporciona AWS credenciales que puede usar para interactuar con AWS los servicios de sus componentes personalizados.

El servicio de intercambio de fichas se ejecuta como un servidor local. Este servidor local se conecta al proveedor de AWS IoT credenciales mediante el alias de AWS IoT rol que usted configura en el componente core core de Greengrass. El componente proporciona dos variables de entorno, `AWS_CONTAINER_CREDENTIALS_FULL_URI` y `AWS_CONTAINER_AUTHORIZATION_TOKEN`. `AWS_CONTAINER_CREDENTIALS_FULL_URI` define el URI de este servidor local. Cuando un componente crea un cliente AWS SDK, el cliente reconoce esta variable de entorno URI y utiliza el token que contiene `AWS_CONTAINER_AUTHORIZATION_TOKEN` para conectarse al servicio de intercambio de fichas y recuperar AWS las credenciales. Esto permite que los dispositivos principales de Greengrass llamen a las operaciones de servicio de AWS . Para obtener más información sobre cómo utilizar este componente en componentes personalizados, consulte Interacción con servicios de AWS.

> **Important**
>
> El 13 de julio de 2016 se agregó el AWS SDKs soporte para adquirir AWS credenciales de esta manera. Su componente debe usar una versión AWS del SDK que se haya creado en esa fecha o después. Para obtener más información, consulte Uso de un AWS SDK compatible en la Guía para desarrolladores de Amazon Elastic Container Service.

> **Note**
>
> Los componentes que utilizan la cadena de credenciales AWS predeterminada pueden descubrir y utilizar otras fuentes de credenciales, si existen. Dentro del Amazon Elastic Container Service (Amazon ECS), las credenciales de

contenedor prevalecerán `AWS_CONTAINER_CREDENTIALS_FULL_URI` sobre `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` las credenciales, lo que significa que las credenciales del servicio de intercambio de fichas pueden no funcionar en Amazon Elastic Container Service (Amazon ECS). Para obtener más información, consulte los proveedores de credenciales estandarizados

Temas

- Versiones
- Tipo
- Sistema operativo
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios

Versiones

Este componente tiene las siguientes versiones:

- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Dependencias

Este componente no tiene ninguna dependencia.

Configuración

Este componente ofrece los siguientes parámetros de configuración que puede personalizar cuando implemente el componente.

`port`

Puerto que se utilizará para las conexiones del servicio de intercambio de token. El servicio de intercambio de tokens se reiniciará tras los cambios en la configuración del puerto.

`credentialRetryInSec`

Especifica los intervalos de reintento en segundos cuando Token Exchange Service detecta errores en la solicitud de credenciales.

`clientError`

El intervalo de reintento en segundos para los errores del cliente (4 xx códigos de estado HTTP).

Valor predeterminado: `120`

Valores válidos: de `10` a `42900`

`serverError`

El intervalo de reintento en segundos para los errores del servidor (5 xx códigos de estado HTTP).

Valor predeterminado: `60`

Valores válidos: de `10` a `42900`

`unknownError`

El intervalo de reintento en segundos para errores desconocidos (errores de conexión y códigos de estado HTTP fuera de los rangos 4xx y 5xx).

Valor predeterminado: `300`

Valores válidos: de `10` a `42900`

Example Ejemplo: actualización de la combinación de configuraciones

```
{
   "port": 2020,
   "credentialRetryInSec": {
    "clientError": 30,
    "serverError": 45,
    "unknownError": 60
   }
}
```

Archivo de registro local

Este componente utiliza el mismo archivo de registro que el componente núcleo de Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la AWS IoT Greengrass carpeta raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
2.0.3	Versión inicial.

Colector IoT SiteWise OPC UA

El componente recopilador SiteWise OPC UA de IoT (`aws.iot.SiteWiseEdgeCollectorOpcua`) permite que AWS IoT SiteWise las pasarelas recopilen datos de los servidores OPC UA locales.

Con este componente, AWS IoT SiteWise las pasarelas se pueden conectar a varios servidores OPC UA. Para obtener más información sobre AWS IoT SiteWise las puertas de enlace, consulte Uso AWS IoT SiteWise en el borde en la Guía del AWS IoT SiteWise usuario.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada
- Datos de salida
- Archivo de registro local
- Licencias
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 3.1.x
- 3.0.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal de Greengrass debe ejecutarse en una de las siguientes plataformas:
  - sistema operativo: Ubuntu 20.04 o posterior

    arquitectura: x86_64 (AMD64) o (Aarch64) ARMv8
  - sistema operativo: Red Hat Enterprise Linux (RHEL) 8

    arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8
  - sistema operativo: Amazon Linux 2

arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8

- sistema operativo: Debian 11

  arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8

- sistema operativo: Windows Server 2019 o posterior

  arquitectura: x86_64 () AMD64

- El dispositivo principal de Greengrass debe permitir la conectividad de red saliente a los servidores OPC UA.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

En la tabla siguiente, se muestran las dependencias de todas las versiones de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.3.0 <3.0.0	Rígido
Administrador de flujos	>2.0.10<3.0.0	Rígido
Administrador de secretos	>=2.0.8 <3.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente no tiene ningún parámetro de configuración.

Puede usar la AWS IoT SiteWise consola o la API para configurar el componente recopilador IoT SiteWise OPC UA. Para obtener más información, consulte el Paso 4: Agregar orígenes de datos (opcional) en la Guía del usuario de AWS IoT SiteWise .

Datos de entrada

Este componente solo acepta datos en los siguientes formatos; todos los demás se ignorarán y descartarán. La siguiente tabla asigna los tipos de datos del OPC UA a sus SiteWise equivalentes.

SiteWise data type	Tipo de datos OPC UA	Descripción

`STRING`	`String`
	`Guid`
	`XmlElement`
		Una cadena con una longitud máxima de 1024 bytes.

`INTEGER`	`SByte`
	`Byte`
	`Int16`
	`UInt16`
	`Int32`
	`UInt32*`
	`Int64*`
		Un entero de 32 bits firmado con un intervalo de `-2,147,483,648 to 2,147,483,647` .

`DOUBLE`	`UInt32*`
	`Int64*`
	`Float`
	`Double`
		Un número de coma flotante con un intervalo de `-10^100 to 10^100` y precisión doble `IEEE 754`.

`BOOLEAN`	`Boolean`	`true` o bien `false`.

* Para los tipos de datos OPC UA `UInt32` y`Int64`, su tipo de SiteWise datos será `INTEGER` si SiteWise es capaz de representar su valor; de lo contrario, lo será. `DOUBLE`

Datos de salida

Este componente escribe `BatchPutAssetPropertyValue` mensajes en el administrador de AWS IoT Greengrass transmisiones. Para obtener más información, consulta BatchPutAssetPropertyValue en la AWS IoT SiteWise Referencia de la API de .

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
   10 -Wait
  ```

Licencias

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
3.1.0	Nuevas características • Agrega la capacidad de configuración del proceso de descubrimiento de nodos para cada origen de datos. • Agrega la capacidad de configuración del activador de cambios de datos predeterminado para cada origen de datos. Mejoras y correcciones de errores • Soluciona las vulnerabilidades de seguridad.
3.0.3	Mejoras y correcciones de errores • Soluciona un problema que provocaba el truncamiento de los archivos y un error de sincronización al sincronizar configuraciones de gran tamaño. • Soluciona un problema que provocaba retrasos en el proceso de inicio debido a la latencia de la suscripción de AWS IoT Core . • Soluciona un problema que provocaba una detección completa tras una desconexión momentánea de los servidores OPC UA.
3.0.2	Mejoras y correcciones de errores • Soluciona la métrica `OpcUaCollector.IncomingValuesCount` para incluir con precisión los puntos de instantáneas en los cálculos.
3.0.1	Mejoras y correcciones de errores • Soluciona las vulnerabilidades de seguridad. • Soluciona un problema con el manejo de la marca de tiempo OPC UA. Agrega una selección de marcas de tiempo preferidas y mejora la lógica alternativa cuando no están disponibles.

Versión	Cambios
	• Soluciona un problema que impedía que las configuraciones del recopilador OPC UA y las transmisiones clásicas y la puerta de enlace V2 pasaran de forma predeterminada al administrador de flujo cuando no se especificaba ningún destino.
3.0.0	Nuevas características Se ha añadido compatibilidad con las pasarelas V3 habilitadas para MQTT, además de las pasarelas V2 clásicas (conocidas anteriormente como pasarelas Edge autohospedadas). SiteWise Para obtener más información, consulte Puertas de enlace V3 habilitadas para MQTT para Edge. AWS IoT SiteWise • Permite publicar los datos del OPC UA en el broker MQTT 5 (AWS IoT Greengrass EMQX). Mejoras y correcciones de errores • Soluciona las vulnerabilidades de seguridad.
2.6.0	Nuevas características • Se agregó soporte para la ingesta de valores Null y NaN si está habilitad a en AWS IoT SiteWise. Para ver o modificar la configuración de Null y NaN en AWS IoT SiteWise, consulte DescribeStorageConfigurationy PutStorageConfiguration APIs.
2.5.1	Mejoras y correcciones de errores • Soluciona un error que cancelaba las futuras tareas de creación de instantáneas si se producía un error mientras se ejecutaba una tarea de instantáneas. • Soluciona un error que provocaba que las actualizaciones de configura ción del origen de datos persistieran hasta después de reiniciar el recopilador OPC UA si se perdía la conexión con el servidor OPC UA del origen de datos.

Versión	Cambios
2.5.0	Nuevas características • Añade una opción de fuente de datos para convertir matrices y DateTime valores simples en cadenas. • Agrega una opción de grupo de propiedades para seleccionar una marca temporal de origen o servidor al recopilar datos de un servidor OPC UA. Mejoras y correcciones de errores • Restablece la configuración de sondeo predeterminada para usar la marca temporal de origen.
2.4.2	Mejoras y correcciones de errores • Soluciona problemas durante la detección del servidor OPC UA en los que se podía detector un nodo varias veces. • Corrige las características de instantáneas para garantizar que la marca temporal sea nueva para cada punto de datos de la instantánea.
2.4.1	Mejoras y correcciones de errores • Soluciona problemas relacionados con la compatibilidad con el proxy. • Soluciona el problema por el que la limpieza de subprocesos fallaba y provocaba un bloqueo de datos.
2.4.0	Nuevas características • Agrega un registro de eventos para facilitar la identificación y la solución de los problemas. Mejoras y correcciones de errores • Soluciona un problema en el cliente OPC UA que provocaba errores en los certificados al conectarse a un servidor OPC UA que utilizaba la versión 1.05 de la especificación OPC UA.

Versión	Cambios
2.3.0	Nuevas características • Suma compatibilidad con la configuración del proxy HTTP del núcleo de Greengrass en Linux. Mejoras y correcciones de errores • Corrige un problema de seguridad (CVE-2019-19135).
2.2.0	Nuevas características • Añade compatibilidad con la instalación del paquete de recopilación de datos en la ARMv8 arquitectura Linux. • Requisitos mínimos para Linux ARMv8: • Memoria: 4 GB • CPU: ARM Cortex-A72 o especificación equivalente Mejoras y correcciones de errores • Mejora el registro de las métricas en el proceso de detección de nodos. • Mejora el manejo de los tipos de datos no compatibles. • Mejora el registro de los errores del flujo de datos.
2.1.3	Nuevas características • Agrega compatibilidad con Windows Server 2019 o posterior. Mejoras y correcciones de errores • Mejora los mensajes de error al implementar este componente en dispositivos no compatibles.

Versión	Cambios

2.1.1	Nuevas características
	• Suma compatibilidad para configurar las siguientes propiedades de suscripción:
	  • DataChangeTrigger- Puede definir la condición que inicia una alerta de cambio de datos.
	  • QueueSize- La profundidad de la cola en un servidor OPC-UA para una métrica concreta en la que se ponen en cola las notificaciones de los elementos monitorizados.
	  • PublishingIntervalMilliseconds- El intervalo (en milisegundos) de un ciclo de publicación especificado al crear una suscripción.
	  • SnapshotFrequencyMilliseconds - Puede configurar el tiempo de espera de la frecuencia de las instantáneas para garantizar que AWS IoT SiteWise Edge ingiera un flujo constante de datos.
	• Esta versión admite la ingesta de datos de calidad `BAD` y filtra los datos en función de las siguientes calidades de datos:
	  • Calidad de datos `UNCERTAIN`
	  • Calidad de datos `BAD`
	Mejoras y correcciones de errores
	• Mejoras en las métricas de los clientes.
	• Corrige la codificación de seguridad que a veces causaba problemas al conectarse a servidores con el cifrado activado.
	• Soluciona un problema por el que el grupo de propiedades no se podía actualizar.

2.0.3	Mejoras y correcciones de errores
2.0.2	Se corrigieron errores y se mejoró la sincronización de prioridad de los activos con periferia.
2.0.1	Versión inicial.

Véase también

- ¿Qué es? AWS IoT SiteWise en la Guía AWS IoT SiteWise del usuario.

Simulador de fuente de datos IoT SiteWise OPC UA

El componente simulador de fuente de datos SiteWise OPC UA de IoT (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) inicia un servidor OPC UA local que genera datos de muestra. Utilice este servidor OPC UA para simular una fuente de datos leída por el componente recopilador SiteWise OPC UA de IoT en una AWS IoT SiteWise puerta de enlace. A continuación, puede explorar las AWS IoT SiteWise características con estos datos de ejemplo. Para obtener más información sobre AWS IoT SiteWise las puertas de enlace, consulte Uso AWS IoT SiteWise en el borde en la Guía del AWS IoT SiteWise usuario.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 1.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal de Greengrass debe poder utilizar el puerto 4840 del host local. El servidor OPC UA local de este componente se ejecuta en este puerto.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

En la tabla siguiente, se muestran las dependencias de todas las versiones de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.3.0 <3.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
  ```

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
1.0.0	Versión inicial.

Versión	Cambios
	Agrega compatibilidad con Windows Server 2016 o posterior.

Véase también

- ¿Qué es AWS IoT SiteWise? en la Guía AWS IoT SiteWise del usuario.

SiteWise Publicador de IoT

El componente SiteWise publicador de IoT (`aws.iot.SiteWiseEdgePublisher`) permite a AWS IoT SiteWise las pasarelas exportar datos del borde al Nube de AWS.

Para obtener más información sobre AWS IoT SiteWise las puertas de enlace, consulte Uso AWS IoT SiteWise en el borde en la Guía del AWS IoT SiteWise usuario.

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Datos de entrada
- Archivo de registro local
- Solución de problemas y depuración
- Licencias
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 4.1.x
- 4.0.x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal de Greengrass debe ejecutarse en una de las siguientes plataformas:
  - sistema operativo: Ubuntu 18.04 o posterior

    arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8
  - sistema operativo: Red Hat Enterprise Linux (RHEL) 8

arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8

- sistema operativo: Amazon Linux 2

  arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8

- sistema operativo: Debian 11

  arquitectura: x86_64 () o (Aarch64) AMD64 ARMv8

- sistema operativo: Windows Server 2019 o posterior

  arquitectura: x86_64 () AMD64

- El dispositivo principal de Greengrass debe estar conectado a Internet.
- El dispositivo principal de Greengrass debe estar autorizado para realizar la acción `iotsitewise:BatchPutAssetPropertyValue`. Para obtener más información, consulte Autorizar los dispositivos principales para que interactúen con los servicios. AWS

  Example política de permisos

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        }
    ]
}
```

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Sí	Publique datos en AWS IoT SiteWise.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.x a 2.2.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Núcleo de Greengrass	>=2.3.0<3.0.0	Rígido
Administrador de flujos	>=2.0.10<3.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente no tiene ningún parámetro de configuración.

Puede usar la AWS IoT SiteWise consola o la API para configurar el componente de SiteWise editor de IoT. Para obtener más información, consulte el Paso 3: Configurar el publicador (opcional) en la Guía del usuario de AWS IoT SiteWise .

Datos de entrada

Este componente lee `PutAssetPropertyValueEntry` los mensajes del administrador de AWS IoT Greengrass transmisiones. Para obtener más información, consulta PutAssetPropertyValueEntry en la AWS IoT SiteWise Referencia de la API de .

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
  Wait
  ```

Solución de problemas y depuración

Este componente incluye un registro de nuevos eventos para ayudar a los clientes a identificar y solucionar problemas. El archivo de registro es independiente del archivo de registro local y se encuentra en la siguiente ubicación. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs
\IotSiteWisePublisherEvents.log
```

Este registro incluye información detallada e instrucciones de solución de errores. Se proporciona información sobre solución de problemas, junto con los diagnósticos, una descripción de cómo solucionar el problema y, a veces, enlaces a más información. La información de diagnóstico incluye lo siguiente:

- Nivel de gravedad
- Timestamp
- Información adicional específica del evento

Example Registro de ejemplo

```
accountBeingThrottled:
  Summary: Data upload speed slowed due to quota limits
  Level: WARN
  Timestamp: '2023-06-09T21:30:24.654Z'
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points
 ingested"
    quota for a customers account. See the associated documentation and associated
    metric for the number of requests that were limited for more information. Note
    that this may be temporary and not require any change, although if the issue
 continues
```

```
    you may need to request an increase for the mentioned quota.
  FurtherInformation:
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
  AssociatedMetrics:
  - Name: TotalErrorCount
    Description: The total number of errors of this type that occurred.
    Value: 327724.0
  AssociatedData:
  - Name: AggregatePropertyAliases
    Description: The aggregated property aliases of the throttled data.
    FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

Licencias

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
4.1.4	Mejoras y correcciones de errores • El SiteWise editor de IoT ahora informa las métricas del estado del sistema, como el uso de memoria y CPU del SiteWise editor de IoT, a CloudWatch. • Se actualizó el informe de errores AWS IoT Greengrass para que el SiteWise editor de IoT se reinicie si el flujo de trabajo de inicio tiene un error parcial.
4.1.3	Mejoras y correcciones de errores • Soluciona una fuga de memoria en el proceso de puntos de control de datos. • Soluciona las vulnerabilidades de seguridad.

Versión	Cambios
4.1.2	Mejoras y correcciones de errores • Soluciona un problema relacionado con la sincronización de grandes configuraciones. • Los retrasos en la suscripción ya AWS IoT Core no afectarán al proceso de puesta en marcha. • Los retrasos en la inicialización de los puntos de control ya no afectarán al proceso de inicio.
4.1.1	Mejoras y correcciones de errores • Soluciona un problema que AWS IoT SiteWise provocaba un uso elevado de la CPU y la memoria en las transmisiones clásicas y las puertas de enlace V2 al enviar datos a través de un destino almacenado en búfer mediante Amazon S3 tras reiniciar el editor. • Agrega soporte para anular el comportamiento de registro de componentes predeterminado mediante la creación de un archivo opcional ubicado en: `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/log4j2.xml` .
4.1.0	Nuevas características • Agrega una herramienta de línea de comandos para administrar los usuarios del agente EMQX MQTT integrado. Esta interfaz permite administrar las cuentas de usuario y configura las políticas de control de acceso para los permisos de publicación y suscripción de temas de MQTT. Mejoras y correcciones de errores • Soluciona el cálculo que determina cuántos alias únicos pueden caber en un único archivo Parquet de ingestión almacenado en búfer.

Versión	Cambios
4.0.3	Mejoras y correcciones de errores • Actualiza el máximo de alias únicos por archivo de ingesta AWS IoT SiteWise almacenado en búfer para que coincida con el límite de 10 000 en la nube. • Corrige un problema de AWS IoT Greengrass registro que informaba incorrectamente de los estados de servicio no válidos. • Agrega validación para evitar la duplicación de nombres de flujos en las configuraciones de destino de Amazon S3 en transmisiones clásicas y puertas de enlace V2. • Corrige un problema de compatibilidad relacionado con la ingesta de AWS IoT SiteWise búfer en particiones no estándar. AWS • Soluciona un problema que provocaba la ingesta de datos duplicada.
4.0.2	Mejoras y correcciones de errores • Soluciona un problema que impedía la carga de datos una vez transcurr ido el tiempo de espera del lote. • Agrega la validación para verificar que cada destino en las configura ciones habilitadas en la puerta de enlace de la versión 3 para MQTT sea único.
4.0.1	Mejoras y correcciones de errores • Soluciona un problema que impedía la carga de datos AWS IoT SiteWise al deshabilitar la configuración de tiempo máximo de espera por lotes. • Soluciona un problema que provocaba que la métrica `IoTSiteWi sePublisher.NumberOfSubscriptionsToMqttBroker` indicara de forma correcta el número de temas únicos suscritos al agente de MQTT.

Versión	Cambios
4.0.0	Nuevas características Añade compatibilidad con las pasarelas V3 habilitadas para MQTT, además de las pasarelas clásicas y V2 (anteriormente denominadas pasarelas Edge autohospedadas). SiteWise • Añade la posibilidad de suscribirse y recibir datos del bróker MQTT 5 (EMQX). AWS IoT Greengrass Para obtener más información, consulte Puertas de enlace V3 habilitadas para MQTT para Edge. AWS IoT SiteWise • Agrega opciones de configuración para los destinos de datos mediante filtros de ruta. Para obtener más información, consulte Comprender los filtros de ruta para los destinos de Edge. AWS IoT SiteWise Mejoras y correcciones de errores • Soluciona un problema donde la métrica `PublishSuccessCount` indicaba valores negativos. • Soluciona un problema que provocaba que el publicador no se iniciara en 100 segundos y pasara al estado `BROKEN`. • Soluciona las vulnerabilidades de seguridad.
3.2.0	Nuevas características • Se agregó soporte para la ingesta de valores Null y NaN si está habilitad a en AWS IoT SiteWise. Para ver o modificar la configuración de Null y NaN en AWS IoT SiteWise, consulte DescribeStorageConfigurationy PutStorageConfiguration APIs. Mejoras y correcciones de errores • Soluciona problemas que causan la corrupción de los archivos de bases de datos de puntos de control. • Soluciona problemas al generar métricas duplicadas.

Versión	Cambios
3.1.4	**Warning** La versión 3.1.4 se discontinuó el 20 de febrero de 2025. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Soluciona problemas que podían provocar tiempos de longer-than-expected inicio después de estar sin conexión.

3.1.3	Mejoras y correcciones de errores
	• Soluciona un problema que impedía que se registraran eventos incluso cuando se creaba el archivo de registro de eventos ubicado en `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/IoTSiteWisePublisherEvents.log` .
	• Añade las siguientes CloudWatch métricas para supervisar la conexión con el broker MQTT:
	  • `IoTSiteWisePublisher.IsConnectedToMqttBroker`
	  • `IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker`
	  • `IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived`
	  • `IoTSiteWisePublisher.MqttMessageReceivedSuccessCount`
	  • `IoTSiteWisePublisher.MqttReceivedSuccessBytes`
	
	Para obtener más información sobre estas métricas, consulte puertas de enlace de AWS IoT Greengrass Version 2.
	• Soluciona un problema que provocaba que se siguiera llamando a la API `BatchCreateJob` aunque se produjera un error al cargar un archivo parquet a S3.

Versión	Cambios
3.1.2	Mejoras y correcciones de errores • Soluciona el problema del uso elevado de la CPU ingresado en la versión 3.1.1.
3.1.1	**Warning** La versión 3.1.1 se suspendió el 12 de marzo de 2024. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Agrega un registro adicional que identifica los alias de datos afectados cuando se produce un error. • Añade la aplicación local de los límites de la AWS IoT SiteWise API a la antigüedad de los datos ingeridos. • Soluciona el problema por el que Publisher confunde los puntos de control de las StreamManager transmisiones cuando hay varios destinos de Amazon S3. • Corrige un obstáculo en el rendimiento relacionado con la forma en que el editor lee las transmisiones. StreamManager
3.1.0	Nuevas características • Suma soporte para publicar datos como archivos parquet en Amazon S3. • Añade compatibilidad con la ingesta en AWS IoT SiteWise búfer.
3.0.0	Mejoras y correcciones de errores • Soluciona problemas relacionados con la compatibilidad con el proxy. Nuevas características • Permite la ingesta de datos desde un agente de MQTT.

Versión	Cambios
2.4.1	Mejoras y correcciones de errores • Habilita el componente para que funcione con Java Corretto 11 versión 11.0.20.8.1 y más reciente. Las versiones 2.4.0 y 2.3.3 del componente muestran el mensaje de error `"Could not find or load main class"` cuando se utilizan con Java Corretto versión 11.0.20.8.1.
2.4.0	Nuevas características • Agrega un nuevo registro de eventos para facilitar la identificación y la solución de los problemas. Mejoras y correcciones de errores • Mejora la recuperación de los puntos de control del publicador.
2.3.3	Mejoras y correcciones de errores • Mejora la capacidad para soportar un alto rendimiento.
2.3.2	Mejoras y correcciones de errores • Corrige la compatibilidad con el proxy HTTP al descargar la configura ción del publicador.

2.3.1	Nuevas características
	• Añade compatibilidad con la instalación del paquete de recopilación de datos en la arquitectura Linux. ARMv8
	• Requisitos mínimos para Linux ARMv8:
	  • Memoria: 4 GB
	  • CPU: ARM Cortex-A72 o especificación equivalente

2.2.3	Mejoras y correcciones de errores • Elimina el reintento de una excepción genérica que no estaba en la lista de excepciones recuperables.

Versión	Cambios
2.2.2	Mejoras y correcciones de errores • Reintroduce la compatibilidad con la carga de datos a AWS IoT SiteWise través de un servidor proxy HTTP.
2.2.1	**Note** Esta versión no admite la configuración de proxy HTTP. La versión 2.2.2 y más recientes reintroducen compatibilidad con esta característica. Nuevas características • Suma compatibilidad con este componente para activar o desactivar la compresión al cargar datos en AWS IoT SiteWise.

Versión	Cambios

2.2.0	**Note**
	Esta versión no admite la configuración de proxy HTTP. La versión 2.2.2 y más recientes reintroducen compatibilidad con esta característica.
	
	Nuevas características
	• Actualiza este componente para comprimir los datos antes de enviarlos al servicio de AWS IoT SiteWise .
	  • En la mayoría de los casos, este cambio reduce el uso del ancho de banda en un 75 % en comparación con las versiones anteriores de este componente.
	  • En la mayoría de los casos, este cambio aumenta el uso de la CPU hasta un 5 %. En las puertas de enlace que procesan grandes cantidades de datos, este cambio puede aumentar el uso de la CPU hasta en un 15 %.
	  • Este cambio no afecta a los cargos por AWS IoT SiteWise servicio ni al uso de la cuota de servicio.
	  • Agrega compatibilidad con Windows Server 2019 o posterior.
	Mejoras y correcciones de errores
	• Corrige un problema que impedía que este componente se iniciara cuando el archivo de puntos de control estaba dañado.

2.1.4	Mejoras y correcciones de errores • Corrige la compatibilidad con la versión 8 de Java.

Versión	Cambios
2.1.3	**Warning** Esta versión ya no está disponible, excepto en las regiones EE.UU. Este (Ohio), Canadá (Central) AWS GovCloud y (EE.UU. Este). Esta versión de componente requiere la versión 11 o posterior de Java para ejecutarse. Las mejoras de esta versión están disponibles en versiones posteriores de este componente. Mejoras y correcciones de errores • Mejora los mensajes de error al implementar este componente en dispositivos no compatibles. • Actualiza los errores de registro cuando se produce un error al cargar los datos.
2.1.2	Mejoras y correcciones de errores • Se actualiza para invocar la característica de exportación de datos caducados tan pronto como los datos caduquen.
2.1.1	Mejoras y correcciones de errores
2.1.0	Nuevas características • Suma compatibilidad para publicar primero los datos más recientes en la nube. • Suma compatibilidad para no publicar datos caducados en la nube. • Suma compatibilidad con el almacenamiento local de datos caducados. Mejoras y correcciones de errores • Reduce el disco I/O y la latencia correspondiente.
2.0.2	Mejoras y correcciones de errores
2.0.1	Versión inicial.

Véase también

- ¿Qué es AWS IoT SiteWise? en la Guía AWS IoT SiteWise del usuario.

SiteWise Procesador IoT

El componente de SiteWise procesador de IoT (`aws.iot.SiteWiseEdgeProcessor`) permite que las pasarelas AWS IoT SiteWise Classic Streams V2 procesen datos en la periferia.

Con este componente, AWS IoT SiteWise las pasarelas pueden utilizar modelos de activos y activos para procesar datos en los dispositivos de pasarela. Para obtener más información sobre AWS IoT SiteWise las puertas de enlace, consulte Uso AWS IoT SiteWise en el borde en la Guía del AWS IoT SiteWise usuario.

> **Note**
>
> La característica del paquete de procesamiento de datos (DPP) dejará de estar abierta a nuevos clientes a partir del 7 de noviembre de 2025. Si desea utilizar el DPP, regístrese antes de esa fecha. Los clientes existentes pueden seguir utilizando el servicio con normalidad. Para obtener más información, consulte Data processing pack availability change en la Guía del usuario de AWS IoT SiteWise .

Temas

- Versiones
- Tipo
- Sistema operativo
- Requisitos
- Dependencias
- Configuración
- Archivo de registro local
- Licencias
- Registros de cambios
- Véase también

Versiones

Este componente tiene las siguientes versiones:

- 3.5.x
- 3.4.x
- 3.3.x
- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente es un componente genérico (`aws.greengrass.generic`). El núcleo de Greengrass ejecuta los scripts del ciclo de vida del componente.

Para obtener más información, consulte Tipos de componentes.

Sistema operativo

Este componente se puede instalar en los dispositivos principales que ejecutan los siguientes sistemas operativos:

- Linux
- Windows

Requisitos

Este componente tiene los siguientes requisitos:

- El dispositivo principal de Greengrass debe ejecutarse en una de las siguientes plataformas:
  - sistema operativo: Ubuntu 20.04 o posterior

arquitectura: x86_64 () AMD64

- sistema operativo: Red Hat Enterprise Linux (RHEL) 8

  arquitectura: x86_64 () AMD64

- sistema operativo: Amazon Linux 2

  arquitectura: x86_64 () AMD64

- sistema operativo: Windows Server 2019 o posterior

  arquitectura: x86_64 () AMD64

- sistema operativo: Debian 11 (Bullseye) o posterior

  arquitectura: x86_64 () AMD64

- El dispositivo principal de Greengrass debe permitir tráfico entrante en el puerto 443.
- El dispositivo principal de Greengrass debe permitir tráfico saliente en los puertos 443 y 8883.
- Los siguientes puertos están reservados para su uso por AWS IoT SiteWise: 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8086, 8445, 9000, 9500, 11080 y 50010. El uso de un puerto reservado para el tráfico puede causar la terminación de la conexión.

  > **Note**
  >
  > El puerto 8087 solo es necesario para la versión 2.0.15 y versiones posteriores de este componente.

- El rol de dispositivo de Greengrass debe tener permisos que le permitan usar AWS IoT SiteWise puertas de enlace en sus dispositivos. AWS IoT Greengrass V2 Para obtener más información, consulte Requisitos en la Guía del usuario de AWS IoT SiteWise .

Puntos de conexión y puertos

Este componente debe poder realizar solicitudes salientes a los siguientes puntos de conexión y puertos, además de a los puntos de conexión y puertos necesarios para el funcionamiento básico. Para obtener más información, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`model.iotsitewise.region.amazonaws.com`	443	Sí	Obtenga información sobre sus AWS IoT SiteWise activos y modelos de activos.
`edge.iotsitewise.region.amazonaws.com`	443	Sí	Obtenga información sobre la configuración de la AWS IoT SiteWise puerta de enlace del dispositivo principal.
`ecr.region.amazonaws.com`	443	Sí	Descargue las imágenes de Docker de AWS IoT SiteWise Edge Gateway desde Amazon

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			Elastic Container Registry.
`iot.region.amazonaws.com`	443	Sí	Obtenga puntos de conexión de dispositivos para su Cuenta de AWS.
`sts.region.amazonaws.com`	443	Sí	Obtenga el ID de su. Cuenta de AWS
`monitor.iotsitewise. region.amazonaws.com`	443	No	Obligatorio si accedes a los AWS IoT SiteWise Monitor portales desde el dispositivo principal.

Dependencias

Al implementar un componente, AWS IoT Greengrass también despliega versiones compatibles de sus dependencias. Esto significa que debe cumplir los requisitos del componente y de todas sus dependencias para poder implementar el componente correctamente. En esta sección, se

enumeran las dependencias de las versiones publicadas de este componente y las restricciones de las versiones semánticas que definen las versiones de los componentes para cada dependencia. También puede ver las dependencias de cada versión del componente en la consola de AWS IoT Greengrass. En la página de detalles del componente, busque la lista de Dependencias.

En la siguiente tabla, se muestran las dependencias de las versiones 2.0.x a 2.1.x de este componente.

Dependencia	Versiones compatibles	Tipo de dependencia
Servicio de intercambio de token	>=2.0.3 <3.0.0	Rígido
Administrador de flujos	>=2.0.10 <3.0.0	Rígido
CLI de Greengrass	>=2.3.0 <3.0.0	Rígido

Para obtener más información sobre las dependencias del componente, consulte la referencia de receta de componentes.

Configuración

Este componente no tiene ningún parámetro de configuración.

Archivo de registro local

Este componente usa el siguiente archivo de registro.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

Visualización de los registros de este componente

- Ejecute el siguiente comando en el dispositivo de núcleo para ver el archivo de registro de este componente en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta AWS IoT Greengrass raíz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -
  Wait
  ```

Licencias

Este componente incluye las siguientes licencias o software de terceros:

- Apache 2.0
- MIT
- Cláusula BSD-2
- Cláusula BSD-3
- CDDL-1.0
- CDDL-1.1
- ISC
- Zlib
- GPL-3.0 con excepción de la GCC
- Dominio público
- Python 2.0
- Unicode-DFS-2015
- Cláusula BSD-1
- OpenSSL
- EPL-1.0

- EPL-2.0
- GPL-2.0- with-classpath-exception
- MPL-2.0
- CC0-1.0
- JSON

Este conector se publica en el Contrato de Licencia de Software de Greengrass Core.

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del componente.

Versión	Cambios
3.5.1	Nuevas características Se agregó soporte para la ingesta de valores Null y NaN si está habilitada en AWS IoT SiteWise. Para ver o modificar la configuración de Null y NaN en AWS IoT SiteWise, consulte DescribeStorageConfigurationy PutStorageConfiguration APIs. Mejoras y correcciones de errores Se actualizaron las dependencias para abordar posibles vulnerabilidades de seguridad.
3.4.0	Nuevas características • Se agregó la compatibilidad opcional con proxy HTTP y HTTPS para permitir la comunicación mediante puertas de enlace con servidores proxy que requieren certificados personalizados. Para obtener más información sobre la configuración de un proxy HTTP para el AWS IoT Greengrass núcleo, consulteRealizar la conexión en el puerto 443 o a través de un proxy de red. Para obtener más información sobre el soporte de proxy específico de SiteWise Edge, consulte Administrar almacenes de confianza para obtener soporte de proxy de AWS IoT SiteWise Edge en la Guía del AWS IoT SiteWise usuario. • Se agregaron ajustes configurables de tiempo de espera de sesión para administrar los períodos de inactividad de AWS OpsHub Edge. SiteWise

Versión	Cambios
	APIs Para obtener más información, consulte Configurar los tiempos de espera de sesión para AWS IoT SiteWise Edge en la Guía del AWS IoT SiteWise usuario. Mejoras en el rendimiento Se redujo el tiempo que tardan los datos entrantes en llegar al almacenamiento del dispositivo de periferia de 5 segundos a menos de 1 segundo. La latencia para la carga de datos AWS IoT SiteWise permanece inalterada.
3.3.1	Nueva característica • Se agregó la compatibilidad opcional con CORS a SiteWise Edge APIs, lo que mejora las capacidades de intercambio de recursos entre orígenes. Esta función mejora la flexibilidad de las aplicaciones web que interactúan con. APIs
3.3.0	Mejoras en el rendimiento • Mecanismo de actualización de la caché optimizado para reducir el I/O uso entre las sincronizaciones de AWS IoT SiteWise activos, ya que solo actualiza las entradas de los activos nuevos o actualizados. • Menor consumo de memoria para mantener una caché con un gran número de propiedades de activos sincronizados. Mejoras y correcciones de errores • Se han suprimido los registros para incorporar valores de propiedades individuales cuando no hay errores de ingesta, lo que reduce el ruido de los registros cuando hay altas tasas de ingesta. • Se mejoró la legibilidad de los registros mediante el uso de un formato legible para las personas en determinadas entradas de registro. • Se agregó compatibilidad con Java 17 y versiones posteriores.

Versión	Cambios
3.2.1	Mejoras y correcciones de errores • Se solucionó el problema por el que las llamadas a la AWS IoT SiteWise API no se paginaban de forma sincrónica con Edge. SiteWise • Se solucionó el problema por el que ya no se publicaba la métrica `MessageRemaining.SiteWise_Edge_Stream` . • Se agregaron las siguientes CloudWatch métricas para monitorear la conexión con el broker MQTT. • `IoTSiteWiseProcessor.IsConnectedToMqttBroker` • `IoTSiteWiseProcessor.NumberOfSubscriptionsToMqttBroker` • `IoTSiteWiseProcessor.NumberOfUniqueMqttTopicsReceived` • `IoTSiteWiseProcessor.MqttMessageReceivedSuccessCount` • `IoTSiteWiseProcessor.MqttReceivedSuccessBytes` Para obtener más información sobre estas métricas, consulte puertas de enlace de AWS IoT Greengrass Version 2.

Versión	Cambios
3.2.0	Mejoras en el rendimiento • Optimice los servicios de API para que ocupen menos memoria y requieran menos espacio en disco para su instalación • Esto proporciona una reducción de 2 GB en el uso de memoria inicial (ahora utiliza 7,5 GB de memoria al arranque, aunque se recomiend an 16 GB) y una reducción de 500 MB en el tamaño de la descarga (ahora requiere una descarga de 1,4 GB) para todo el componente. Nuevas características • La API `GetAssetPropertyValueAggregates` ahora admite ventanas de agregación de 15 minutos en la periferia. • Los puertos 8081 y 8082 ya no necesitan estar disponibles para que este componente se ejecute correctamente. **Note** El punto final local del plano de AWS IoT SiteWise datos APIs, por ejemplo`get-asset-property-value` , se está cambiando de `http://localhost:8081` a`http://localhost:11080/data` . El punto final local del plano de AWS IoT SiteWise control APIs, por ejemplo`list-asset-models` , se cambia de `http://localhost:11080` a`http://localhost:11080/control` . AWS siempre recomienda utilizar los puntos de enlace HTTPS de la puerta de enlace SiteWise Edge. Esos puntos de conexión no han cambiado. Mejoras y correcciones de errores • La sincronización desde ahora AWS IoT SiteWise hará que los recursos pasen a un estado válido si se interrumpió la sincronización anterior. Esto solucionará los problemas relacionados con la corrupción de algunos recursos tras un reinicio forzado. • Corrige un problema poco frecuente en el que un recurso puede dañarse en la periferia si se modifica durante la sincronización. La

Versión	Cambios
	sincronización ahora fallará si se detecta esta condición y el recurso se volverá a intentar en la siguiente sincronización. • Corrige un problema que podría haber permitido llamar externamente APIs al punto final HTTP. Ahora solo se puede usar HTTPS para llamar APIs fuera de la dirección de bucle invertido local. • La API `ListAssets` ahora muestra las jerarquías de activos de los activos almacenados en la periferia. • Soluciona un problema que provocaba que el paquete de procesami ento de datos no se pudiera reiniciar, actualizar o revertir en Windows. • Corrige un error en el paquete de procesamiento de datos para el sistema operativo Windows que impedía a los clientes utilizar las credenciales para conectarse con un agente de MQTT.
3.1.3	Mejoras y correcciones de errores • Se solucionó el problema por el que el paquete de procesamiento de datos informaba incorrectamente de una sincronización exitosa cuando algunos de los recursos fallaban. • Permita que varios activos tengan el mismo nombre siempre que no tengan el mismo elemento principal.
3.1.1	Mejoras y correcciones de errores • Se ha solucionado el problema por el que la solicitud de SiGv4 fallaba debido a una discordancia entre zonas horarias. • Se solucionó el problema por el que las propiedades de transform ación y métrica dejaban de calcularse cuando se basaban en atributos después de reiniciarse. • Habilite la compatibilidad con la configuración de puertos personalizada del administrador de flujos. • Soluciona un problema por el que las propiedades que están sincroniz adas con la periferia podrían dejar de actualizarse.

Versión	Cambios
3.1.0	Mejoras y correcciones de errores • Se ha solucionado el problema por el que la API `ListAssetModels` no podía generar el siguiente token.
3.0.0	Nuevas características • Permite la ingesta de datos desde un agente de MQTT.
2.2.1	Mejoras y correcciones de errores • Ajuste el proceso de sincronización para que el almacenamiento de datos en el plano de control sea más coherente con el funcionamiento de la nube. Esto afecta levemente a la actualización. **Note** Los datos del plano de control sincronizados en la versión 2.2.1 o posterior no serán compatibles con las versiones anteriores. Para cambiar a versiones anteriores, debe realizar una instalación nueva. Esto no afecta a las actualizaciones. Los datos sincronizados en las versiones anteriores funcionarán con la versión 2.2.1. • Modificaciones adicionales en la cadena de AWS credenciales para AWS IoT Greengrass V2 priorizar las credenciales.
2.1.37	Mejoras y correcciones de errores • Elimine el dependency-routing-service proceso e incorpore su funcionalidad al property-state-service proceso para reducir el uso de recursos de los procesos que se comunican. • Aumente el límite máximo de resultados de la `get-asset-property-value-history` API a 20 000 para que coincida con el límite utilizado por AWS IoT SiteWise. • Se solucionó un problema por el que el siguiente token no aparecía en los resultados paginados de la API `get-asset-property-value-history` cuando no se especificaba un límite máximo de resultados.

Versión	Cambios
2.1.35	Mejoras y correcciones de errores • Modifica la cadena de AWS credenciales para priorizar las credenciales. AWS IoT Greengrass • Soluciona un problema relacionado con la detección de cuentas al implementarlas como parte de un grupo de objetos AWS IoT .
2.1.34	Mejoras y correcciones de errores • Ajusta los metric/transform cálculos para utilizar subprocesos múltiples en Linux. Windows sigue ejecutando cálculos de un solo subproceso para garantizar la compatibilidad. • Corrige un problema por el que faltaban cálculos métricos en algunas ventanas de cálculo.
2.1.33	Mejoras y correcciones de errores • Corrige un problema con los informes de estado de error a la consola de Greengrass.
2.1.32	Mejoras y correcciones de errores • Suma compatibilidad con nombres de usuario y grupos personalizados.
2.1.31	Mejoras y correcciones de errores • Suma compatibilidad para calcular el promedio ponderado en el tiempo y la desviación estándar ponderada en el tiempo para los datos modelados en AWS IoT SiteWise.
2.1.29	Mejoras y correcciones de errores • Agrega la funcionalidad de filtrado de activos en la periferia.
2.1.28	Mejoras y correcciones de errores • Optimiza la sincronización de los recursos para permitir que una gran cantidad de activos se sincronicen desde la periferia Nube de AWS hasta la periferia.

Versión	Cambios
2.1.24	Mejoras y correcciones de errores • Soluciona un problema que provocaba que el panel desapareciera al sincronizar un recurso por segunda vez.
2.1.23	Mejoras y correcciones de errores • Se agregó un tiempo de espera para el proceso de instalación `aws.iot.SiteWiseEdgeProcessor` a fin de evitar errores en la instalación si la conectividad a Internet es lenta. • Sincronización de recursos optimizada para mejorar la eficiencia de la sincronización entre la nube y la periferia.
2.1.21	**Warning** La actualización de la versión 2.0.x a la versión 2.1.x provocará la pérdida de datos locales. Nuevas características • Agrega compatibilidad con Windows Server 2019 o posterior. • Elimina el docker para los sistemas operativos basados en Linux.
2.0.16	Esta versión contiene correcciones de errores y mejoras.
2.0.15	Mejoras y correcciones de errores • Cambia el puerto que utiliza este componente para las operaciones de la API de sincronización de recursos del 8085 al 8087. Como resultado , este componente ahora requiere que el puerto 8087 esté disponible. Este componente aún requiere que el puerto 8085 esté disponible. • Actualiza la AWS OpsHub autenticación para denegar a los usuarios no autorizados durante el inicio de sesión, en lugar de cuando un usuario intenta llamar a las operaciones de la API.
2.0.14	Esta versión contiene correcciones de errores y mejoras.

Versión	Cambios
2.0.13	Mejoras y correcciones de errores • Soluciona un problema por el que, cuando este componente reporta datos a CloudWatch las métricas de Amazon, ahora indica correctam ente qué datos no están modelados.
2.0.9	Mejoras y correcciones de errores • Mejora la fiabilidad a la hora de crear y actualizar AWS IoT SiteWise recursos en el dispositivo principal. • Agrega operaciones de API locales adicionales que puede usar para monitorear qué componentes están instalados en el dispositivo principal , la versión de cada componente y el estado de cada componente. Puede ver esta información en la pestaña Configuración de la AWS IoT SiteWise aplicación AWS OpsHub correspondiente al dispositivo principal. • Agrega un estado a los contenedores de Docker en los que se ejecuta este componente. Puede ejecutar el comando `docker ps` para ver el estado de los contenedores.
2.0.7	Mejoras y correcciones de errores • Corrige la compatibilidad con la visualización de AWS IoT SiteWise Monitor portales en el dispositivo principal.
2.0.6	Mejoras y correcciones de errores • Corrige las `latest()` funciones AWS IoT SiteWise `statetime () earliest()` , y que este componente calcula en el dispositivo principal.
2.0.5	Mejoras y correcciones de errores • Añade compatibilidad con la AWS IoT SiteWise `pretrigger()` función en las transformaciones que este componente calcula en el dispositivo principal. • Cambia la ruta en la que este componente almacena la configuración del Protocolo ligero de acceso a directorios (LDAP) para la autentica ción.

Versión	Cambios
2.0.2	Versión inicial.

Véase también

- ¿Qué es? AWS IoT SiteWise en la Guía AWS IoT SiteWise del usuario.

Componentes compatibles con Publisher

Los componentes compatibles con Publisher se encuentran en versión preliminar para AWS IoT Greengrass y están sujetos a cambios. Estos componentes no son compatibles con AWS. Debe ponerse en contacto con Publisher si tiene algún problema con cada uno de los componentes.

Los componentes de Greengrass compatibles con Publisher son desarrollados, ofrecidos y mantenidos por proveedores de componentes externos. Los proveedores de componentes externos provienen del catálogo de dispositivos de AWS Partner, AWS Heroes o proveedores comunitarios. Para comprar los componentes de este catálogo, póngase en contacto directamente con el proveedor de componentes externo.

Entre los componentes de Greengrass que se admiten en Publisher se incluyen los siguientes:

Temas

- AiShield.edge
- Sensor AI de EdgeLabs
- Greengrass S3 Ingestor

AiShield.edge

Este componente fue desarrollado y es compatible con AiShield, con tecnología Bosch. Aumente la seguridad de su IA con AIShield.edge. Este componente está diseñado para implementar sin problemas defensas personalizadas e informadas para las amenazas en los dispositivos de periferia, lo que protege sus dispositivos contra los ataques de la IA.

Este método ofrece las siguientes ventajas:

- Puede pasar sin problemas del análisis de vulnerabilidades con AiShield AI Security a las defensas de periferia reforzadas dentro de AWS
- Puede implementar defensas personalizadas en varios dispositivos de periferia con facilidad
- Puede ampliar la protección adaptada a diversas configuraciones de IA que admite varios tipos de modelos y marcos
- Puede mantenerse actualizado con una integración perfecta en los flujos de trabajo de Amazon SageMaker AI y Greengrass
- Puede obtener información inmediata sobre las posibles amenazas, con los datos transmitidos directamente a AWS IoT Core
- Puede obtener una ruta de seguridad de IA cohesiva para la implementación de defensa en la periferia de AIShield AI Security en AWS Marketplace

Este componente debe ejecutarse en una de las siguientes plataformas:

- sistema operativo: Linux

Si está interesado en adquirir este componente, póngase en contacto con Bosch Software and Digital Solutions: `<AIShield.Contact@bosch.com>`.

Sensor AI de EdgeLabs

Este componente fue desarrollado y es compatible con AI EdgeLabs. El sensor AI de EdgeLabs es una aplicación basada en contenedores que contiene capacidades de detección y prevención de amenazas basadas en la IA. Sensor AI está integrado en un componente de Greengrass y se implementa como un contenedor independiente en el dispositivo principal junto con otros componentes de Greengrass.

Este componente actual es un agente basado en contenedores que verifica continuamente la comunicación de la red y busca patrones de amenazas en el software que se ejecuta en el Edge Host o en la puerta de enlace de IoT. Este componente utiliza eBPF, la verificación del comportamiento del ancho de banda de los procesos y la configuración basada en el host. La funcionalidad principal de este componente se basa en las funciones NDR/IPS y EDR.

Este método ofrece las siguientes ventajas:

- Detección de amenazas basada en IA contra ataques de red y malware (EDR/NDR)
- Respuesta a incidentes (IPS) automatizada basada en la IA

- Inteligencia de amenazas local en el servidor con una transferencia de datos externa mínima
- Implementación ligera con Docker y Greengrass

Este componente debe ejecutarse en una de las siguientes plataformas:

- sistema operativo: Linux

Si está interesado en comprar este componente, póngase en contacto con AI EdgeLabs: `<contact@edgelabs.ai.>`

Greengrass S3 Ingestor

Este componente fue desarrollado y cuenta con soporte de Nathan Glover. El componente Greengrass S3 Ingestor está diseñado para usarse con el componente administrador de flujos. Este componente toma un flujo de mensajes JSON delimitado por líneas del administrador de flujos y los agrupa en un archivo GZIP. Este componente permite la ingesta eficiente de datos en Amazon S3 para su posterior procesamiento o almacenamiento. Este componente no admite el envío de datos a la Nube de AWS en tiempo real.

Este componente debe ejecutarse en una de las siguientes plataformas:

- sistema operativo: Linux
- sistema operativo: Windows

Si está interesado en adquirir este componente, póngase en contacto con Nathan Glover: `<nathan@glovers.id.au.>`

Componentes de la comunidad

El catálogo de software de Greengrass es un índice de los componentes de Greengrass desarrollados por la comunidad de Greengrass. Desde este catálogo, puede descargar, modificar e implementar componentes para crear sus aplicaciones de Greengrass. Puede ver el catálogo en el siguiente enlace: https://github.com/aws-greengrass/aws-greengrass-software-catalog.

Cada componente tiene un repositorio público de GitHub que puede explorar. Consulte el catálogo de software de Greengrass en GitHub para encontrar la lista completa de los componentes de la comunidad. Por ejemplo, este catálogo incluye los siguientes componentes:

- Amazon Kinesis Video Streams

  Este componente ingiere flujos de audio y video de cámaras locales que utilizan el Protocolo de transmisión en tiempo real (RTSP). A continuación, el componente carga los flujos de audio y video a Amazon Kinesis Video Streams.

- Puerta de enlace IoT Bluetooth

  Este componente utiliza la biblioteca BluePy, que permite la comunicación con dispositivos Bluetooth de bajo consumo (LE) para crear interfaces de cliente Bluetooth LE.

- Rotador de certificados

  Este componente proporciona un medio para rotar el certificado del dispositivo principal de AWS IoT Greengrass y la clave privada, a toda la flota y a gran escala.

- Túneles seguros en contenedores

  Este componente proporciona un contenedor de Docker para una tunelización segura con todas las dependencias y bibliotecas coincidentes en una receta reutilizable que no depende de un sistema operativo host específico.

- Grafana

  Este componente le permite alojar un servidor Grafana en un dispositivo principal de Greengrass. Puede usar los paneles de Grafana para visualizar y administrar los datos en el dispositivo principal.

- GStreamer para Amazon Lookout for Vision

  Este componente proporciona un complemento de GStreamer para que pueda realizar la detección de anomalías de Lookout for Vision en sus canalizaciones de GStreamer personalizadas.

- Asistente doméstico

  Este componente permite al cliente utilizar el Asistente doméstico para controlar localmente los dispositivos domésticos inteligentes. Proporciona integración con los servicios de AWS en la periferia y en la nube para ofrecer soluciones de automatización del hogar que amplían el asistente doméstico.

- Panel de control de InfluxDBGrafana

Este componente proporciona una experiencia de un solo clic para configurar los componentes InfluxDB y Grafana. Conecta InfluxDB a Grafana y automatiza la configuración de un panel local de Grafana que representa la telemetría de AWS IoT Greengrass en tiempo real.

- InfluxDB

  Este componente proporciona una base de datos de serie temporal de InfluxDB en un dispositivo principal de Greengrass. Puede usar este componente para procesar datos de sensores de IoT, analizar datos en tiempo real y monitorear las operaciones en la periferia.

- Publicador InfluxDB

  Este componente transmite la telemetría del estado del sistema AWS IoT Greengrass desde el complemento emisor del núcleo a InfluxDB. Este componente también puede reenviar telemetría personalizada a InfluxDB.

- Marco de publicación/suscripción de IoT

  Este marco proporciona una arquitectura de aplicación, un código de plantilla y ejemplos para implementar que ayudan a mejorar la calidad del código para las aplicaciones de publicación/suscripción de IoT distribuidas y basadas en eventos que utilizan componentes personalizados de AWS IoT Greengrass versión 2. Para obtener más información, consulte Creación de componentes de AWS IoT Greengrass.

- Laboratorios de Jupyter

  Este componente implementa JupyterLab en un dispositivo principal de AWS IoT Greengrass. El entorno de Jupyter tiene acceso a los recursos variables de proceso y entorno establecidos por AWS IoT Greengrass, lo que simplifica el proceso de prueba y desarrollo de componentes escritos en Python.

- Servidor web local

  Este componente le permite crear una interfaz de usuario web local en un dispositivo principal de Greengrass. Puede crear una interfaz de usuario web local que le permita configurar los ajustes del dispositivo y la aplicación o supervisar el dispositivo, por ejemplo.

- Adaptador de protocolo LoRaWAN

  Este componente ingiere datos de dispositivos inalámbricos locales que utilizan el protocolo LoRaWAN, que es un protocolo de red de área amplia (LPWAN) de bajo consumo. El componente le permite analizar los datos y actuar sobre ellos a nivel local sin comunicarse con la nube.

- Modbus TCP

  Este componente recopila datos de los dispositivos locales mediante el protocolo Modbus-TCP y los publica en flujos de datos seleccionados.

- Node-RED

  Este componente instala Node-RED en un dispositivo principal de AWS IoT Greengrass mediante NPM. El componente depende del componente de autenticación de Node-RED, que debe implementarse y configurarse de forma explícita. Puede usar la CLI de Node-RED para Greengrass a fin de implementar flujos de Node-RED en los dispositivos de AWS IoT Greengrass.

- Docker de Node-RED

  Este componente instala Node-RED en el dispositivo principal de AWS IoT Greengrass mediante el contenedor de Docker de Node-RED oficial. El componente depende del componente de autenticación de Node-RED, que debe implementarse y configurarse de forma explícita. Puede usar la CLI de Node-RED para Greengrass a fin de implementar flujos de Node-RED en los dispositivos de AWS IoT Greengrass.

- Autenticación Node-RED

  Este componente configura un nombre de usuario y una contraseña para proteger la instancia de Node-RED que se ejecuta en un dispositivo principal de AWS IoT Greengrass.

- Enrutador fronterizo OpenThread

  Este componente implementa el contenedor de Docker del enrutador fronterizo OpenThread. El componente ayuda a crear un dispositivo Matter que incluye un enrutador fronterizo Thread.

- Conector de transmisión de datos OSI Pi

  Este componente permite transmitir la ingesta de datos en tiempo real desde el archivo de datos OSI Pi a una arquitectura de datos moderna en AWS. Se integra a OSI Pi Asset Framework, que se administra de forma centralizada a través de la mensajería PubSub de AWS IoT.

- Proveedor de Parsec

  Este componente permite a los dispositivos de AWS IoT Greengrass integrar soluciones de seguridad de hardware mediante el proyecto Parsec de código abierto de Fundación para la computación nativa en la nube (CNCF).

- Base de datos PostgreSQL

Este componente ofrece compatibilidad con la base de datos relacional PostgreSQL en la periferia. Los clientes pueden usar este componente para aprovisionar y administrar una instancia local de PostgreSQL dentro de un contenedor de Docker.

- Cargador de archivos de S3

  Este componente supervisa un directorio en busca de nuevos archivos, los carga en Amazon Simple Storage Service (Amazon S3) y los elimina cuando se carga correctamente.

- Cliente del administrador de secretos

  Este componente proporciona una herramienta de CLI que pueden utilizar otros componentes que necesiten recuperar secretos del componente Administrador de secretos en un script de ciclo de vida de recetas.

- Enrutamiento TES al contenedor

  Este componente configura nftables o iptables en un dispositivo de AWS IoT Greengrass para que pueda usar el componente Servicio de intercambio de token con contenedores.

- WebRTC

  Este componente ingiere los flujos de audio y video de las cámaras RTSP conectadas al dispositivo principal de AWS IoT Greengrass. A continuación, el componente convierte los flujos de audio y video en comunicación punto a punto o en retransmisión a través de Amazon Kinesis Video Streams.

Para solicitar una característica o informar de un error, abra un problema de GitHub en el repositorio de ese componente. AWS no es compatible con los componentes de la comunidad. Para obtener más información, consulte el archivo CONTRIBUTING.md del repositorio de cada componente.

Varios de los componentes proporcionados por AWS también son de código abierto. Para obtener más información, consulte Software AWS IoT Greengrass Core de código abierto.

Herramientas de desarrollo de AWS IoT Greengrass

Utilice las herramientas de desarrollo de AWS IoT Greengrass para crear, probar, compilar, publicar e implementar componentes personalizados de Greengrass.

- CLI del kit de desarrollo de Greengrass

Utilice la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) en el entorno de desarrollo local para crear componentes a partir de plantillas y componentes de la comunidad en el catálogo de software de Greengrass. Puede usar la CLI del GDK para crear el componente y publicarlo en el servicio de AWS IoT Greengrass como un componente privado en su Cuenta de AWS.

- Interfaz de la línea de comandos de Greengrass

  Utilice la interfaz de la línea de comandos de Greengrass (CLI de Greengrass) en los dispositivos principales de Greengrass para implementar y depurar los componentes de Greengrass. La CLI de Greengrass es un componente que puede implementar en los dispositivos principales para crear implementaciones locales, ver detalles sobre los componentes instalados y explorar los archivos de registro.

- Consola de depuración local

  Utilice la consola de depuración local de los dispositivos principales de Greengrass para implementar y depurar los componentes de Greengrass mediante una interfaz web de panel local. La consola de depuración local es un componente que puede implementar en los dispositivos principales para crear implementaciones locales y ver detalles sobre los componentes instalados.

AWS IoT Greengrass también proporciona los siguientes SDK que puede utilizar en los componentes personalizados de Greengrass:

- El SDK para dispositivos con AWS IoT, que contiene la biblioteca de comunicación entre procesos (IPC). Para obtener más información, consulte Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core.
- El SDK del administrador de flujos, que puede utilizar para transferir flujos de datos a la Nube de AWS. Para obtener más información, consulte Administración de flujos de datos en los dispositivos principales de Greengrass.

Temas

- Interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass
- Interfaz de la línea de comandos de Greengrass
- Utilice el marco AWS IoT Greengrass de pruebas

Interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass

La interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) proporciona características que lo ayudan a desarrollar componentes personalizados de Greengrass. Puede usar la CLI del GDK para crear, compilar y publicar componentes personalizados. Al crear un repositorio de componentes con la CLI del GDK, puede partir de una plantilla o un componente de comunidad del catálogo de software de Greengrass. A continuación, puede elegir un sistema de compilación que agrupe los archivos como archivos ZIP, utilice un script de compilación de Maven o Gradle o ejecute un comando de compilación personalizado. Luego de crear un componente, puede usar la CLI del GDK para publicarlo en el servicio de AWS IoT Greengrass, de modo que puede usar la consola de AWS IoT Greengrass o la API para implementar el componente en los dispositivos principales de Greengrass.

Al desarrollar componentes de Greengrass sin la CLI del GDK, debe actualizar los URI de versión y artefacto del archivo de recetas del componente cada vez que cree una nueva versión del componente. Al utilizar la CLI del GDK, esta puede actualizar automáticamente los URI de versión y artefacto cada vez que publique una nueva versión del componente.

La CLI del GDK es de código abierto y está disponible en GitHub. Puede personalizar y ampliar la CLI del GDK para lograr los objetivos de desarrollo de componentes. Lo invitamos a consultar por incidencias y solicitudes de cambios en el repositorio de GitHub. Puede encontrar el origen de la CLI del GDK en el siguiente enlace: https://github.com/aws-greengrass/aws-greengrass-gdk-cli.

Requisitos previos

Para instalar y usar la CLI del kit de desarrollo de Greengrass, necesita lo siguiente:

- Un Cuenta de AWS. Si no dispone de una, consulte Configura un Cuenta de AWS.
- Una computadora de desarrollo similar a Windows, macOS o Unix con conexión a Internet.
- Para la versión 1.1.0 de la CLI del GDK o versiones posteriores, debe tener Python 3.6 instalada en la computadora de desarrollo.

  Para la versión 1.0.0 de la CLI del GDK, debe tener Python 3.8 instalada en la computadora de desarrollo.
- Git instalado en su computadora de desarrollo.
- AWS Command Line Interface (AWS CLI) instalado y configurado con credenciales en su computadora de desarrollo. Para obtener más información, consulte Instalar, actualizar y

desinstalar la AWS CLI y Configurar la AWS CLI en la Guía del usuario de AWS Command Line Interface.

> **Note**
>
> Si usa una Raspberry Pi u otro dispositivo ARM de 32 bits, instale la versión 1 de AWS CLI. AWS CLI La versión 2 no está disponible para dispositivos ARM de 32 bits. Para obtener más información, consulte Instalar, actualizar y desinstalar la versión 1 de AWS CLI.

- Para usar la CLI del GDK para publicar componentes en el servicio de AWS IoT Greengrass, debe contar con los siguientes permisos:
  - `s3:CreateBucket`
  - `s3:GetBucketLocation`
  - `s3:PutObject`
  - `greengrass:CreateComponentVersion`
  - `greengrass:ListComponentVersions`
- Para usar la CLI de GDK para crear un componente cuyos artefactos existan en un bucket de S3 y no en el sistema de archivos local, debe tener los siguientes permisos:
  - `s3:ListBucket`

  Esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

Registro de cambios

En la siguiente tabla, se describen los cambios en cada versión de la CLI del GDK. Para obtener más información, consulte la página de lanzamientos de la CLI del GDK en GitHub.

Versión	Cambios
1.6.2	Mejoras y correcciones de errores • Soluciona el problema por el que el archivo gradlew.bat de Windows no funcionaba debido a la ruta relativa. • Mejoras mínimas en el registro, las pruebas y el empaquetado.

Versión	Cambios
1.6.1	Mejoras y correcciones de errores • Agrega una corrección de seguridad para el análisis de argumentos de la CLI. • Permite al GDK obtener el nombre de la versión más reciente de Greengrass Testing Framework (GTF) como versión GTF predeterm inada. • Permite a GDK recomendar a los clientes que utilizan una versión anterior de GTF que la actualicen a la versión más reciente.
1.6.0	Nuevas características • Agrega una comprobación de validación de receta con respecto al esquema de receta de Greengrass durante los comandos `component build` y `component publish`. Esta actualización ayuda a los desarrolladores a identificar problemas procesables en sus recetas de componentes en una fase temprana del proceso de creación de los componentes. • Agrega un conjunto de pruebas de confianza a la plantilla que se puede implementar con el comando `test-e2e init`. Este conjunto de pruebas de confianza incluye ocho pruebas genéricas que se pueden utilizar y ampliar para adaptarse a las necesidades básicas de las pruebas de componentes. Mejoras y correcciones de errores • Actualiza la versión predeterminada de Greengrass Testing Framework (GTF) utilizada por el comando `test-e2e` a la versión 1.2.0.
1.5.0	Mejoras y correcciones de errores Actualiza los patrones reconocidos por la opción de compilación `excludes` cuando `build_system` está `zip`. Esta versión ahora reconocerá los patrones globales que coincidan con los nombres de las rutas en función de sus caracteres comodín. Esto permite especificar de forma personalizada los directorios de los que se debe excluir.

Versión	Cambios
1.4.0	Nuevas características • Agrega un nuevo comando `config` que inicia una petición interacti va para modificar los campos de un archivo de configuración del GDK existente. • Modifica los comandos `gdk component build` y `gdk component publish` para comprobar que el tamaño de la receta cumple con los requisitos de Greengrass (<=16 000 bytes) antes de continuar. Mejoras y correcciones de errores • Agrega un registro adicional en el resultado del comando `gdk component build` cuando un error de sintaxis de la receta impide que se complete la compilación para detectarlo. • Cambia el nombre de `otf-options` y `otf-version` a `gtf-optio ns` y `gtf-version` , respectivamente, debido al cambio de nombre de Open Test Framework a Greengrass Testing Framework.
1.3.0	Nuevas características • Agrega un nuevo comando `test-e2e` para permitir las pruebas integrales de los componentes mediante Open Test Framework. • Agrega una nueva opción de configuración, `zip_name`, para admitir nombres de archivos zip configurables con el sistema de compilación zip. • Permite que la propiedad `region` del archivo de configuración del GDK sea opcional. Mejoras y correcciones de errores • Soluciona el problema por el que se crea un nuevo directorio incluso cuando la plantilla o el repositorio especificados no existen al inicializar un proyecto de GDK con el argumento `--name`.

Versión	Cambios
1.2.3	Mejoras y correcciones de errores • Soluciona el problema que provocaba un error en la creación del bucket debido a un control de errores incorrecto. • Soluciona el problema por el que se eliminaban las estructuras de listas de la receta del componente.
1.2.2	Mejoras y correcciones de errores • Las claves de recetas ya no diferencian mayúsculas de minúsculas. • Agrega una comprobación para determinar si hay un bucket en una Región de AWS y si el usuario puede acceder a él antes de crear uno nuevo. Requiere que el usuario cuente con el permiso `GetBucketLocation` . • Soluciona el problema con la palabra clave `excludes` del archivo de configuración de la CLI del GDK.
1.2.1	Mejoras y correcciones de errores • Acepta la Región de AWS Canadá (Centro) (`ca-central-1`) en la entrada de configuración de la región en el archivo `gdk-config.json` . • Soluciona problemas con el argumento `--region` de la CLI del GDK para el comando `publish`.

Versión	Cambios
1.2.0	Nuevas características • Agrega la entrada `options` a la configuración `build` en el archivo de configuración de la CLI del GDK. Permite `excludes` en `options` para excluir ciertos archivos del artefacto zip cuando se utiliza el sistema de compilación `zip`. • Agrega el sistema de compilación `gradlew` para usar Gradle Wrapper para compilar componentes. • Suma compatibilidad con los archivos de compilación DSL de Kotlin para la opción de compilación `gradle`. • Agrega una entrada `options` a la configuración `publish` en el archivo de configuración de la CLI del GDK. Admite `file_upload_args` en `options` para proporcionar argumentos adicionales al cargar archivos en Amazon S3. Mejoras y correcciones de errores • Soluciona el problema por el que las compilaciones de Gradle no se borraban antes de ejecutar un comando de compilación. • Soluciona el problema por el que la compilación no se cerraba cuando se producía un error en el comando de compilación. • Mejora el formato de salida del comando `gdk component list`.

Versión	Cambios
1.1.0	Nuevas características • Agrega compatibilidad con el sistema de compilación de Gradle. • Agrega compatibilidad con el sistema de compilación de Maven en dispositivos Windows. • Agrega el argumento `--bucket` al comando component publish. Puede usar este argumento para especificar el bucket exacto en el que la CLI de GDK carga los artefactos de los componentes. • Agrega el argumento `--name` al comando component init. Puede usar esta opción para especificar la carpeta en la que la CLI del GDK inicializ a el componente. • Suma compatibilidad con los artefactos de los componentes que existen en un bucket de S3, pero no en la carpeta de creación del componente local. Puede utilizar esta característica para reducir los costos de ancho de banda de los artefactos de componentes de gran tamaño, como los modelos de machine learning. Mejoras y correcciones de errores • Actualiza el comando component publish para comprobar si el componente está creado antes de publicarlo. Si el componente no está creado, este comando ahora lo crea automáticamente. • Soluciona el problema por el que el sistema de compilación zip no se puede compilar en dispositivos Windows cuando el nombre del archivo ZIP contiene letras mayúsculas. • Mejora el formato de los mensajes de registro y cambia el nivel de registro predeterminado a `INFO` en los dispositivos que ejecutan versiones de Python anteriores a la versión 3.8. • Cambia el requisito mínimo de la versión de Python a 3.6.
1.0.0	Versión inicial.

Instalación o actualización de la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass

La interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) se basa en Python, por lo que puede utilizar `pip` para instalarla en su computadora de desarrollo.

> **Tip**
>
> También puede instalar la CLI del GDK en un entorno virtual de Python, como venv. Para obtener más información, consulte Entornos y paquetes virtuales en la documentación de Python 3.

Cómo instalar o actualizar la CLI del GDK

1. Ejecute el siguiente comando para instalar la versión más reciente de la CLI del GDK desde el repositorio de GitHub.

   ```
   python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
   ```

   > **Note**
   >
   > Para instalar una versión específica de la CLI del GDK, sustituya *versionTag* por la etiqueta de versión que desee instalar. Puede ver las etiquetas de versión de la CLI del GDK en su repositorio de GitHub.
   >
   > ```
   > python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
   > ```

2. Ejecute los siguientes comandos para verificar que la CLI del GDK se instaló correctamente.

   ```
   gdk --help
   ```

   Si no se encuentra el comando `gdk`, agregue su carpeta a PATH.

   - En dispositivos Linux, agregue `/home/MyUser/.local/bin` a PATH y sustituya *MyUser* por el nombre de su usuario.

- En dispositivos Windows, agregue *PythonPath*`\\Scripts` a PATH y sustituya *PythonPath* por la ruta a la carpeta Python de su dispositivo.

Ahora puede usar la CLI del GDK para crear, compilar y publicar componentes de Greengrass. Para obtener más información acerca de cómo utilizar la CLI del GDK, consulte Comandos de la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass.

Comandos de la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass

La interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) proporciona una interfaz de línea de comandos que puede utilizar para crear, compilar y publicar componentes de Greengrass en la computadora de desarrollo. Los comandos de la CLI del GDK utilizan el siguiente formato.

```
gdk <command> <subcommand> [arguments]
```

Al instalar la CLI del GDK, el instalador agrega `gdk` a la PATH para que pueda ejecutar la CLI del GDK desde la línea de comandos.

Puede utilizar los siguientes argumentos con cualquier comando:

- Utilice `-h` o `--help` para obtener información sobre un comando de la CLI del GDK.
- Utilice `-v` o `--version` para ver qué versión de la CLI del GDK está instalada.
- Utilice `-d` o `--debug` para generar registros detallados que pueda usar para depurar la CLI del GDK.

En esta sección, se describen los comandos de la CLI del GDK y se proporcionan ejemplos para cada comando. La sinopsis de cada comando muestra sus argumentos y cómo se utilizan. Los argumentos opcionales deben ir entre corchetes.

Comandos disponibles

- componente
- config
- test-e2e

componente

Utilice el comando `component` en la interfaz de línea de comandos del kit de desarrollo (CLI del GDK) AWS IoT Greengrass para crear, compilar y publicar componentes personalizados de Greengrass.

Subcomandos

- init
- build
- publish
- list

init

Inicialice una carpeta de componentes de Greengrass desde una plantilla de componentes o un componente de la comunidad.

La CLI del GDK recupera los componentes de la comunidad del catálogo de software de Greengrass y las plantillas de componentes del repositorio de plantillas de componentes de AWS IoT Greengrass en GitHub.

> **Note**
>
> Si usa la versión 1.0.0 de la CLI de GDK, debe ejecutar este comando en una carpeta vacía. La CLI de GDK descarga la plantilla o el componente de comunidad en la carpeta actual.
> Si usa la versión 1.1.0 de la CLI de GDK o una versión posterior, puede especificar el argumento `--name` para especificar la carpeta en la que la CLI de GDK descarga la plantilla o el componente de comunidad. Si usa este argumento, especifique una carpeta que no existe. La CLI de GDK crea la carpeta por usted. Si no se especifica este argumento, la CLI de GDK usa la carpeta actual, que debe estar vacía.
> Si el componente usa el sistema de compilación zip, la CLI del GDK comprime ciertos archivos de la carpeta del componente en un archivo zip con el mismo nombre que la carpeta del componente. Por ejemplo, si el nombre de la carpeta del componente es `HelloWorld`, la CLI de GDK crea un archivo zip denominado `HelloWorld.zip`. En la receta del componente, el nombre del artefacto zip debe coincidir con el nombre de la carpeta del componente. Si usa la versión 1.0.0 de la CLI de GDK en un dispositivo Windows, los nombres de las carpetas y los archivos zip de los componentes deben contener solo letras minúsculas.

Si inicializa una plantilla o un componente de comunidad que utiliza el sistema de compilación zip en una carpeta con un nombre diferente al de la plantilla o el componente, debe cambiar el nombre del artefacto zip en la receta del componente. Actualice las definiciones `Artifacts` y `Lifecycle` de forma que el nombre del archivo zip coincida con el nombre de la carpeta del componente. Los siguientes ejemplos resaltan el nombre del archivo zip en las definiciones `Artifacts` y `Lifecycle`.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
```

```
    Lifecycle:
      Run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
{configuration:/Message}"
```

Sinopsis

```
$ gdk component init
    [--language]
    [--template]
    [--repository]
    [--name]
```

Argumentos (se inicializan desde la plantilla del componente)

- `-l`, `--language`: el lenguaje de programación que se utilizará para la plantilla que especifique.

  Debe especificar `--repository` o `--language` y `--template`.
- `-t`, `--template`: la plantilla de componentes que se utilizará en un proyecto de componentes locales. Utilice el comando list para ver las plantillas disponibles.

  Debe especificar `--repository` o `--language` y `--template`.
- `-n`, `--name`: (opcional) el nombre de la carpeta local en la que la CLI del GDK inicializa el componente. Especifique una carpeta que no existe. La CLI de GDK crea la carpeta por usted.

  Esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

Argumentos (se inicializan desde el componente de la comunidad)

- `-r`, `--repository`: el componente de la comunidad que se va a incluir en la carpeta local. Para ver los componentes de la comunidad disponibles, utilice el comando list.

  Debe especificar `--repository` o `--language` y `--template`.
- `-n`, `--name`: (opcional) el nombre de la carpeta local en la que la CLI del GDK inicializa el componente. Especifique una carpeta que no existe. La CLI de GDK crea la carpeta por usted.

  Esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

Output

El siguiente ejemplo muestra el resultado que se produce al ejecutar este comando para inicializar una carpeta de componentes desde la plantilla Hello World de Python.

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
 component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
 from Greengrass Software Catalog.
```

El siguiente ejemplo muestra el resultado que se produce al ejecutar este comando para inicializar una carpeta de un componente de la comunidad.

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
 from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

Convierta el código de origen de un componente en una receta y artefactos que pueda publicar en el servicio de AWS IoT Greengrass. La CLI del GDK ejecuta el sistema de compilación que especifique en el archivo de configuración de la CLI del GDK, `gdk-config.json`. Debe ejecutar este comando en la misma carpeta en la que se encuentra el archivo `gdk-config.json`.

Al ejecutar este comando, la CLI del GDK crea una receta y artefactos en la carpeta `greengrass-build` de la carpeta del componente. La CLI del GDK guarda la receta en la carpeta `greengrass-build/recipes` y guarda los artefactos en la carpeta `greengrass-build/artifacts/`*`componentName`*`/`*`componentVersion`*.

Si usa la CLI del GDK versión 1.1.0 o una versión posterior, la receta del componente puede especificar los artefactos que existen en un bucket de S3, pero no en la carpeta de compilación del componente local. Puede utilizar esta característica para reducir el uso del ancho de banda cuando desarrolle componentes con artefactos de gran tamaño, como los modelos de machine learning.

Después de crear un componente, puede realizar una de las siguientes acciones para probarlo en un dispositivo principal de Greengrass:

- Si desarrolla en un dispositivo diferente al que ejecuta el software AWS IoT Greengrass Core, debe publicar el componente para implementarlo en un dispositivo principal de Greengrass. Publique el componente en el servicio de AWS IoT Greengrass e impleméntelo en el dispositivo

principal de Greengrass. Para obtener más información, consulte los comandos publish y Crear implementaciones.

- Si desarrolla el servicio en el mismo dispositivo en el que ejecuta el software AWS IoT Greengrass Core, puede publicar el componente en el servicio de AWS IoT Greengrass que desea implementar o puede crear una implementación local para instalar y ejecutar el componente. Para crear una implementación local, utilice la CLI de Greengrass. Para obtener más información, consulte Interfaz de la línea de comandos de Greengrass y Prueba de los componentes de AWS IoT Greengrass con implementaciones locales. Al crear la implementación local, especifique `greengrass-build/recipes` como carpeta de recetas y `greengrass-build/artifacts` como carpeta de artefactos.

Sinopsis

```
$ gdk component build
```

Argumentos

Ninguno

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

publish

Publique el componente en el servicio de AWS IoT Greengrass. Este comando carga los artefactos de compilación en un bucket de S3, actualiza el URI del artefacto en la receta y crea una nueva versión del componente a partir de la receta. La CLI del GDK utiliza el bucket de S3 y la región de AWS que se especifican en el archivo de configuración de la CLI del GDK, `gdk-config.json`. Debe ejecutar este comando en la misma carpeta en la que se encuentra el archivo `gdk-config.json`.

Si usa la versión 1.1.0 o posteriores de la CLI de GDK, puede especificar el argumento `--bucket` para especificar el bucket de S3 en el que la CLI de GDK carga los artefactos del componente. Si no especifica este argumento, la CLI de GDK se carga en el bucket de S3 cuyo nombre es *`bucket`*`-`*`region`*`-`*`accountId`*, donde *`bucket`* y *`region`* son los valores que especifica en `gdk-config.json` y *`acountId`* es el ID de su Cuenta de AWS. La CLI de GDK crea el bucket si no existe.

Si utiliza la versión 1.2.0 de la CLI del GDK o una versión posterior, puede anular la Región de AWS especificada en el archivo de configuración de la CLI del GDK mediante el parámetro `--region`. También puede especificar opciones adicionales mediante el parámetro `--options`. Para ver la lista de opciones disponibles, consulte Archivo de configuración la CLI del kit de desarrollo de Greengrass.

Al ejecutar este comando, la CLI del GDK publica el componente con la versión que especifique en la receta. Si especifica `NEXT_PATCH`, la CLI del GDK utilizará la siguiente versión de parche que aún no exista. Las versiones semánticas siguen un sistema de números de major.minor.patch. Para obtener más información, consulte la especificación semántica de la versión.

> **Note**
>
> Si utiliza la versión 1.1.0 de la CLI del GDK o una versión posterior, al ejecutar este comando, la CLI del GDK comprueba si el componente está creado. Si el componente no está compilado, la CLI del GDK lo compila antes de publicarlo.

Sinopsis

```
$ gdk component publish
    [--bucket] [--region] [--options]
```

Argumentos

- `-b`, `--bucket`: (opcional) especifique el nombre del bucket de S3 en el que la CLI del GDK publica los artefactos de los componentes.

  Si no especifica este argumento, la CLI de GDK se carga en el bucket de S3 cuyo nombre es *`bucket`*`-`*`region`*`-`*`accountId`*, donde *`bucket`* y *`region`* son los valores que especifica en `gdk-config.json` y *`acountId`* es el ID de su Cuenta de AWS. La CLI de GDK crea el bucket si no existe.

  La CLI de GDK crea el bucket si no existe.

  Esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.
- `-r`, `--region`: (opcional) especifique el nombre de la Región de AWS para cuando el componente se crea. Este argumento anula el nombre de la región en la configuración de la CLI del GDK.

  Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.
- `-o`, `--options` (opcional) Especifique una lista de opciones para publicar un componente. El argumento debe ser una cadena JSON válida o una ruta de archivo a un archivo JSON que contenga las opciones de publicación. Este argumento anula las opciones en la configuración de la CLI del GDK.

  Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:45:30] INFO - No private version of the component
 'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
 version to create.
```

```
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
 If this is your first time using this bucket, add the 's3:GetObject' permission
 to each core device's token exchange role to allow it to download the component
 artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developerguide/device-service-role.html.
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
 com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
 com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
 account.'com.example.PythonHelloWorld'.
```

list

Recupere la lista de plantillas de componentes y componentes de comunidad disponibles.

La CLI del GDK recupera los componentes de la comunidad del catálogo de software de Greengrass y las plantillas de componentes del repositorio de plantillas de componentes de AWS IoT Greengrass en GitHub.

Puede pasar el resultado de este comando al comando init para inicializar los repositorios de componentes a partir de plantillas y componentes de la comunidad.

Sinopsis

```
$ gdk component list
    [--template]
    [--repository]
```

Argumentos

- `-t`, `--template`: (opcional) especifique este argumento para enumerar las plantillas de componentes disponibles. Este comando muestra el nombre y el lenguaje de cada plantilla en el formato *`name`*`-`*`language`*. Por ejemplo, en `HelloWorld-python`, el nombre de la plantilla es `HelloWorld` y el lenguaje es `python`.
- `-r`, `--repository`: (opcional) especifique este argumento para enumerar los repositorios de componentes de la comunidad disponibles.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
 Greengrass Software Catalog.
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.
1. HelloWorld-python
2. HelloWorld-java
```

config

Utilice el comando `config` de la interfaz de la línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) para modificar la configuración del GDK en el archivo de configuración, `gdk-config.json`.

Subcomandos

- actualización

actualización

Inicie una petición interactiva para modificar los campos de un archivo de configuración del GDK existente.

Sinopsis

```
$ gdk config update
    [--component]
```

Argumentos

- `-c`, `--component`: Para actualizar los campos relacionados con los componentes del archivo `gdk-config.json`. Este argumento es obligatorio porque es la única opción.

Output

El siguiente ejemplo muestra el resultado obtenido luego de ejecutar este comando para configurar un componente.

```
$ gdk config update --component
```

```
Current value of the REQUIRED component_name is (default:
 com.example.PythonHelloWorld):
Current value of the REQUIRED author is (default: author):
Current value of the REQUIRED version is (default: NEXT_PATCH):
Do you want to change the build configurations? (y/n)
Do you want to change the publish configurations? (y/n)
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

test-e2e

Utilice el comando `test-e2e` de la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) para inicializar, compilar y ejecutar módulos de prueba integrales en el proyecto del GDK.

Subcomandos

- init
- build
- ejecutar

init

Inicie un proyecto de la CLI del GDK existente con un módulo de pruebas que utiliza Greengrass Testing Framework (GTF).

De forma predeterminada, la CLI del GDK recupera la plantilla del módulo maven del repositorio de plantillas de componentes de AWS IoT Greengrass en GitHub. Este módulo maven viene con una dependencia en el archivo JAR `aws-greengrass-testing-standalone`.

Este comando crea un nuevo directorio llamado `gg-e2e-tests` dentro del proyecto GDK. Si el directorio del módulo de pruebas ya existe y no está vacío, el comando se cierra sin hacer nada. Esta carpeta `gg-e2e-tests` contiene las definiciones de las características y los pasos de Cucumber estructurados en un proyecto de Maven.

Por defecto, este comando intentará usar la última versión de GTF.

Sinopsis

```
$ gdk test-e2e init
```

```
    [--gtf-version]
```

Argumentos

- `-ov`, `--gtf-version`: (opcional) la versión del GTF que se utilizará con el módulo de pruebas integrales del proyecto GDK. Este valor debe ser una de las versiones del GTF de los lanzamientos. Este argumento anula la `gtf_version` en la configuración de la CLI del GDK.

Output

El siguiente ejemplo muestra el resultado que se obtiene al ejecutar este comando para inicializar el proyecto GDK con el módulo de pruebas.

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
 1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
 gg-e2e-tests directory...
```

build

> **Note**
>
> Debe compilar el componente ejecutando gdk component build antes de compilar el módulo de prueba integral.

Creación del módulo de pruebas integral. La CLI del GDK crea el módulo de pruebas mediante el sistema de compilación que especifica en el archivo de configuración de la CLI del GDK, `gdk-config.json`, según la propiedad `test-e2e`. Debe ejecutar este comando en la misma carpeta en la que se encuentra el archivo `gdk-config.json`.

De forma predeterminada, la CLI del GDK usa el sistema de compilación Maven para crear el módulo de pruebas. Se requiere Maven para ejecutar el comando `gdk test-e2e build`.

Debe compilar el componente ejecutando gdk-component-build antes de compilar el módulo de prueba, si los archivos de características de prueba tienen variables como `GDK_COMPONENT_NAME` y `GDK_COMPONENT_RECIPE_FILE` para interpolar.

Al ejecutar este comando, la CLI del GDK interpola todas las variables de la configuración del proyecto de GDK y crea el módulo `gg-e2e-tests` para generar el archivo JAR de prueba final.

Sinopsis

```
$ gdk test-e2e build
```

Argumentos

Ninguno

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.........
```

ejecutar

Ejecute el módulo de pruebas con las opciones de prueba del archivo de configuración del GDK.

> **Note**
>
> Debe compilar el módulo de pruebas ejecutando gdk test-e2e build antes de ejecutar las pruebas integrales.

Sinopsis

```
$ gdk test-e2e run
    [--gtf-options]
```

Argumentos

- `-oo`, `--gtf-options`: (opcional) especifique una lista de opciones para ejecutar las pruebas integrales. El argumento debe ser una cadena JSON válida o una ruta de archivo a un archivo JSON que contenga las opciones de GTF. Las opciones que se proporcionan en el archivo

de configuración se combinan con las que se proporcionan en los argumentos del comando. Si una opción está presente en ambos lugares, la que se encuentra en el argumento tiene prioridad sobre la del archivo de configuración.

Si la opción `tags` no se especifica en este comando, GDK utiliza `Sample` para las etiquetas. Si `ggc-archive` no se especifica, GDK descarga la última versión del archivo del núcleo de Greengrass.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ gdk test-e2e run
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar –ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip –
tags=Sample

16:35:59.693 [] [] [] [INFO]
 com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
 com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.......
```

Archivo de configuración la CLI del kit de desarrollo de Greengrass

La interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI del GDK) lee un archivo de configuración denominado `gdk-config.json` para crear y publicar componentes. Este archivo de configuración debe estar en la raíz del repositorio del componente. Puede utilizar el comando init de la CLI del GDK para inicializar los repositorios de componentes con este archivo de configuración.

Temas

- Formato de archivo de configuración de la CLI del GDK
- Ejemplos de archivo de configuración de la CLI del GDK

Formato de archivo de configuración de la CLI del GDK

Al definir un archivo de configuración de la CLI del GDK para un componente, se especifica la siguiente información en formato JSON.

`gdk_version`

La versión mínima de la CLI del GDK que es compatible con este componente. Este valor debe ser una de las versiones de la CLI del GDK de los lanzamientos.

`component`

La configuración de este componente.

`componentName`

`author`

El autor o publicador del componente.

`version`

Esta es la versión del componente. Especifique uno de los siguientes valores:

- `NEXT_PATCH`: al elegir esta opción, la CLI de GDK establece la versión al publicar el componente. La CLI de GDK consulta el servicio de AWS IoT Greengrass para identificar la versión más reciente publicada del componente. A continuación, establece la versión en la siguiente versión del parche posterior a esa versión. Si no ha publicado el componente antes, la CLI de GDK usa la versión `1.0.0`.

  Si elige esta opción, no podrá usar la CLI de Greengrass para implementar y probar localmente el componente en su computadora de desarrollo local que ejecuta el software AWS IoT Greengrass Core. Para habilitar las implementaciones locales, debe especificar una versión semántica en su lugar.

- Una versión semántica, como **`1.0.0`**. Las versiones semánticas siguen un sistema de números de major.minor.patch. Para obtener más información, consulte la especificación semántica de la versión.

  Si desarrolla componentes en un dispositivo principal de Greengrass en el que desee implementar y probar el componente, elija esta opción. Debe compilar el componente con una versión específica para crear implementaciones locales con la CLI de Greengrass.

`build`

La configuración que se utilizará para convertir el origen de este componente en artefactos. Este objeto contiene la siguiente información:

`build_system`

El sistema de compilación que se va a utilizar. Puede elegir entre las siguientes opciones:

- `zip`: empaqueta la carpeta del componente en un archivo ZIP para definirla como el único artefacto del componente. Elija esta opción para los siguientes tipos de componentes:
  - Componentes que usan lenguajes de programación interpretados, como Python o JavaScript.
  - Componentes que empaquetan archivos distintos del código, como modelos de machine learning u otros recursos.

  La CLI de GDK comprime la carpeta del componente en un archivo zip con el mismo nombre que la carpeta del componente. Por ejemplo, si el nombre de la carpeta del componente es `HelloWorld`, la CLI de GDK crea un archivo zip denominado `HelloWorld.zip`.

  > **Note**
  >
  > Si usa la versión 1.0.0 de la CLI de GDK en un dispositivo Windows, los nombres de las carpetas y los archivos zip de los componentes deben contener solo letras minúsculas.

  Cuando la CLI de GDK comprime la carpeta del componente en un archivo zip, omite los siguientes archivos:

  - El archivo `gdk-config.json`
  - El archivo de receta (`recipe.json` o `recipe.yaml`)
  - Carpetas de compilación, como `greengrass-build`
- `maven`: ejecuta el comando `mvn clean package` para compilar el origen del componente en artefactos. Elija esta opción para los componentes que usan Maven, como los componentes de Java.

En dispositivos Windows, esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

- `gradle`: ejecuta el comando `gradle build` para compilar el origen del componente en artefactos. Elija esta opción para los componentes que usan Gradle. Esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

  El sistema de compilación `gradle` admite Kotlin DSL como archivo de compilación. Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

- `gradlew`: ejecuta el comando `gradlew` para compilar el origen del componente en artefactos. Elija esta opción para los componentes que usan el Gradle Wrapper.

  Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

- `custom`: ejecuta un comando personalizado para compilar el origen del componente en una receta y artefactos. Especifique el comando personalizado en el parámetro `custom_build_command`.

`custom_build_command`

(Opcional) El comando de compilación personalizado que se ejecuta en un sistema de compilación personalizado. Debe especificar este parámetro si especifica `custom` para `build_system`.

> **Important**
>
> Este comando debe crear una receta y artefactos en las siguientes carpetas de la carpeta del componente. La CLI de GDK crea estas carpetas automáticamente al ejecutar el comando component build.
>
> - Carpeta de recetas: `greengrass-build/recipes`
> - Carpeta de artefactos: `greengrass-build/artifacts/componentName/componentVersion`
>
>   Sustituya *componentName* por el nombre del componente y sustituya *componentVersion* por la versión del componente o `NEXT_PATCH`.

Puede especificar una sola cadena o una lista de cadenas, donde cada cadena es una palabra del comando. Por ejemplo, para ejecutar un comando de compilación personalizado para un componente de C++, puede especificar **`cmake --build build --config Release`** o **`["cmake", "--build", "build", "--config", "Release"]`**.

Para ver un ejemplo de un sistema de compilación personalizado, consulte aws.greengrass.labs.LocalWebServer community component en GitHub.

`options`

(Opcional) Se utilizan opciones de configuración adicionales durante el proceso de creación del componente.

Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

`excludes`

Una lista de patrones globales que definen qué archivos se deben excluir del directorio de componentes al crear el archivo zip. Válido solo cuando `build_system` es `zip`.

> **Note**
>
> En las versiones 1.4.0 y anteriores de la CLI del GDK, cualquier archivo que coincida con una entrada de la lista de exclusiones se excluye de todos los subdirectorios del componente. Para lograr el mismo comportamiento en las versiones 1.5.0 y posteriores de la CLI del GDK, anteponga `**/` a las entradas existentes en la lista de exclusiones. Por ejemplo, `*.txt` excluirá los archivos de texto solo del directorio; `**/*.txt` excluirá los archivos de texto de todos los directorios y subdirectorios.
>
> En las versiones 1.5.0 y posteriores de la CLI del GDK, es posible que vea una advertencia durante la compilación del componente cuando `excludes` esté definido en el archivo de configuración del GDK. Para deshabilitar esta advertencia, defina la variable de entorno `GDK_EXCLUDES_WARN_IGNORE` en `true`.

La CLI del GDK siempre excluye los siguientes archivos del archivo zip:

- El archivo `gdk-config.json`
- El archivo de receta (`recipe.json` o `recipe.yaml`)
- Carpetas de compilación, como `greengrass-build`

De forma predeterminada, se excluyen los siguientes archivos. Sin embargo, puede controlar cuáles de estos archivos se excluyen con la opción `excludes`.

- Cualquier carpeta que comience con el prefijo “test” (`test*`)
- Todos los archivos ocultos
- La carpeta `node_modules`

Si especifica la opción `excludes`, la CLI del GDK excluirá solo los archivos que configuró con la opción `excludes`. Si no especifica la opción `excludes`, la CLI del GDK excluye los archivos y carpetas predeterminados indicados anteriormente.

`zip_name`

El nombre del archivo zip que se utilizará al crear un artefacto zip durante el proceso de creación. Válido solo cuando `build_system` es `zip`. Si `build_system` está vacío, el nombre del componente se utiliza como nombre del archivo zip.

`publish`

La configuración que se utilizará para publicar este componente en el servicio de AWS IoT Greengrass.

Si usa la versión 1.1.0 o posteriores de la CLI de GDK, puede especificar el argumento `--bucket` para especificar el bucket de S3 en el que la CLI de GDK carga los artefactos del componente. Si no especifica este argumento, la CLI de GDK se carga en el bucket de S3 cuyo nombre es *`bucket`*`-`*`region`*`-`*`accountId`*, donde *`bucket`* y *`region`* son los valores que especifica en `gdk-config.json` y *`acountId`* es el ID de su Cuenta de AWS. La CLI de GDK crea el bucket si no existe.

Este objeto contiene la siguiente información:

`bucket`

El nombre del bucket de S3 que se utilizará para alojar los artefactos de los componentes.

`region`

La Región de AWS donde la CLI del GDK publica este componente.

Esta propiedad es opcional si utiliza la CLI del GDK versión 1.3.0 o versiones posteriores.

`options`

(Opcional) Se utilizan opciones de configuración adicionales durante la creación de la versión del componente.

Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

`file_upload_args`

Una estructura JSON que contiene argumentos que se envían a Amazon S3 al cargar archivos a un bucket, como metadatos y mecanismos de cifrado. Para obtener una lista de los argumentos permitidos, consulte la clase `S3Transfer` en la documentación de Boto3.

`test-e2e`

(Opcional) La configuración que se utilizará durante las pruebas integrales del componente. Esta característica está disponible para la CLI del GDK de versión 1.3.0 y versiones posteriores.

`build`

`build_system`: el sistema de compilación que se va a utilizar. La opción predeterminada es `maven`. Puede elegir entre las siguientes opciones:

- `maven`: ejecuta el comando `mvn package` para compilar el módulo de pruebas. Elija esta opción para compilar el módulo de pruebas que usa Maven.
- `gradle`: ejecuta el comando `gradle build` para compilar el módulo de pruebas. Elige esta opción para el módulo de pruebas que usa Gradle.

`gtf_version`

(Opcional) La versión del Greengrass Testing Framework (GTF) que se utilizará como dependencia del módulo de pruebas de extremo a extremo al inicializar el proyecto GDK con GTF. Este valor debe ser una de las versiones del GTF de los lanzamientos. El valor predeterminado es la versión 1.1.0 de GTF.

`gtf_options`

(Opcional) Opciones de configuración adicionales utilizadas durante las pruebas integrales del componente.

En la siguiente lista, se incluyen las opciones que puede utilizar con la versión 1.1.0 de GTF.

- `additional-plugins`: (opcional) complementos de Cucumber adicionales
- `aws-region`: apunta a puntos de conexión regionales específicos para los servicios de AWS. El valor predeterminado es lo que detecta el SDK de AWS.
- `credentials-path`: ruta de credenciales de perfil de AWS opcional. El valor predeterminado son las credenciales detectadas en el entorno host.
- `credentials-path-rotation`: duración de rotación opcional para las credenciales de AWS. El valor predeterminado es de 15 minutos o `PT15M`.
- `csr-path`: la ruta de la CSR mediante la cual se generará el certificado del dispositivo.
- `device-mode`: el dispositivo objetivo que se está probando. El valor predeterminado es el dispositivo local.
- `env-stage`: apunta al entorno de implementación de Greengrass. El valor predeterminado es de producción.
- `existing-device-cert-arn`: el ARN de un certificado existente que desee usar como certificado de dispositivo para Greengrass.
- `feature-path`: archivo o directorio que contiene archivos de características adicionales. El valor predeterminado es que no se usan archivos de características adicionales.
- `gg-cli-version`: anula la versión de la CLI de Greengrass. El valor predeterminado es el que se encuentra en `ggc.version`.
- `gg-component-bucket`: el nombre de un bucket de Amazon S3 existente que aloja los componentes de Greengrass.
- `gg-component-overrides`: una lista de anulaciones de componentes de Greengrass.
- `gg-persist`: una lista de elementos de prueba que persisten tras una ejecución de la prueba. El comportamiento predeterminado es no conservar nada. Los valores aceptados son: `aws.resources`, `installed.software` y `generated.files`.
- `gg-runtime`: una lista de valores para influir en la forma en que la prueba interactúa con los recursos de la prueba. Estos valores sustituyen al parámetro `gg.persist`. Si el valor predeterminado está vacío, se asume que todos los recursos de prueba se administran por caso de prueba, incluido el tiempo de ejecución de Greengrass instalado. Los valores aceptados son: `aws.resources`, `installed.software` y `generated.files`.
- `ggc-archive`: la ruta hacia el componente del núcleo de Greengrass archivado.
- `ggc-install-root`: directorio para instalar el componente núcleo de Greengrass. Los valores predeterminados son test.temp.path y la carpeta de ejecución de la prueba.

- `ggc-log-level`: establece el nivel de registro del núcleo de Greengrass para la ejecución de la prueba. El valor predeterminado es “INFO”.
- `ggc-tes-rolename`: el rol de IAM que asumirá AWS IoT Greengrass Core para acceder a los servicios de AWS. Si no existe un rol con un nombre dado, se creará uno con una política de acceso predeterminada.
- `ggc-trusted-plugins`: la lista separada por comas de las rutas (en el host) de los complementos de confianza que deben agregarse a Greengrass. Para indicar la ruta en el propio DUT, agregue el prefijo “dut:” a la ruta
- `ggc-user-name`: el valor user:group posixUser para el núcleo de Greengrass. El valor predeterminado es el nombre de usuario actual con el que se ha iniciado sesión.
- `ggc-version`: anula la versión del componente núcleo de Greengrass en ejecución. El valor predeterminado es el que se encuentra en ggc.archive.
- `log-level`: nivel de registro de la ejecución de la prueba. El valor predeterminado es “INFO”.
- `parallel-config`: conjunto de índices de lotes y número de lotes como cadena JSON. El valor predeterminado del índice de lotes es 0 y el número de lotes es 1.
- `proxy-url`: configura todas las pruebas para enrutar el tráfico a través de esta URL.
- `tags`: ejecuta únicamente etiquetas de características. Se puede intersecar con “&”
- `test-id-prefix`: un prefijo común que se aplica a todos los recursos específicos de la prueba, incluidos los nombres y las etiquetas de los recursos de AWS. El prefijo predeterminado es “gg”.
- `test-log-path`: directorio que contendrá los resultados de toda la ejecución de la prueba. El valor predeterminado es “testResults”.
- `test-results-json`: marcador para determinar si se genera un informe JSON de Cucumber resultante escrito en el disco. El valor predeterminado es true (verdadero).
- `test-results-log`: marcador para determinar si la salida de la consola se genera escrita en el disco. El valor predeterminado es falso.
- `test-results-xml`: marcador para determinar si se genera un informe XML de JUnit resultante escrito en el disco. El valor predeterminado es true (verdadero).
- `test-temp-path`: directorio para generar artefactos de prueba locales. El valor predeterminado es un directorio temporal aleatorio con el prefijo gg-testing.
- `timeout-multiplier`: multiplicador proporcionado para todos los tiempos de espera de las pruebas. El valor predeterminado es 1.0.

Ejemplos de archivo de configuración de la CLI del GDK

Puede hacer referencia a los siguientes ejemplos de archivos de configuración de la CLI del GDK para ayudarlo a configurar los entornos de componentes de Greengrass.

Hello World (Python)

El siguiente archivo de configuración de la CLI del GDK admite un componente Hello world que ejecuta un script de Python. Este archivo de configuración utiliza el sistema de compilación `zip` para empaquetar el script de Python del componente en un archivo ZIP que la CLI del GDK carga como un artefacto.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
           "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
    },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
```

```
  },
  "gdk_version": "1.6.1"
  }
}
```

Hello World (Java)

El siguiente archivo de configuración de la CLI del GDK admite un componente Hello world que ejecuta una aplicación Java. Este archivo de configuración utiliza el sistema de compilación `maven` para empaquetar el código de origen Java del componente en un archivo JAR que la CLI de GDK carga como un artefacto.

```
{
  "component": {
    "com.example.JavaHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
  },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
```

```
}
```

Componentes de la comunidad

Varios componentes de la comunidad del catálogo de software de Greengrass utilizan la CLI del GDK. Puede explorar los archivos de configuración de la CLI del GDK en los repositorios de estos componentes.

Cómo ver los archivos de configuración de la CLI del GDK de los componentes de la comunidad

1. Ejecute el siguiente comando para mostrar en una lista los componentes de la comunidad que utilizan la CLI del GDK.

   ```
   gdk component list --repository
   ```

   La respuesta muestra el nombre del repositorio de GitHub para cada componente de la comunidad que usa la CLI del GDK. Cada repositorio existe en la organización `awslabs`.

   ```
   [2022-02-22 17:27:31] INFO - Listing all the available component repositories from
    Greengrass Software Catalog.
   [2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
   1. aws-greengrass-labs-database-influxdb
   2. aws-greengrass-labs-telemetry-influxdbpublisher
   3. aws-greengrass-labs-dashboard-grafana
   4. aws-greengrass-labs-dashboard-influxdb-grafana
   5. aws-greengrass-labs-local-web-server
   6. aws-greengrass-labs-lookoutvision-gstreamer
   ```

2. Abra el repositorio de GitHub de un componente de la comunidad en la siguiente URL. Sustituya *`community-component-name`* por el nombre de un componente de la comunidad del paso anterior.

   ```
   https://github.com/awslabs/community-component-name
   ```

Interfaz de la línea de comandos de Greengrass

La interfaz de la línea de comandos (CLI) de Greengrass le permite interactuar con AWS IoT Greengrass Core en el dispositivo para desarrollar componentes y depurar problemas de forma local.

Por ejemplo, puede usar la CLI de Greengrass para crear una implementación local y reiniciar un componente en el dispositivo principal.

Implemente el componente CLI de Greengrass (`aws.greengrass.Cli`) para instalar la CLI de Greengrass en el dispositivo principal.

> **Important**
>
> Se recomienda usar este componente solo en entornos de desarrollo y no en entornos de producción. Este componente brinda acceso a información y operaciones que, por lo general, no necesitará en un entorno de producción. Siga el principio de privilegio mínimo al implementar este componente solo en los dispositivos principales donde lo necesite.

Temas

- Instalación de la CLI de Greengrass
- Comandos de la CLI de Greengrass

Instalación de la CLI de Greengrass

Puede instalar la CLI de Greengrass de una de las siguientes maneras:

- Utilice `--deploy-dev-tools` este argumento la primera vez que configure el software AWS IoT Greengrass Core en su dispositivo. También debe especificar `--provision true` si desea aplicar este argumento.
- Implemente el componente de la CLI de Greengrass (`aws.greengrass.Cli`) en su dispositivo.

En esta sección, se describen los pasos para implementar el componente de la CLI de Greengrass. Para obtener información acerca de la instalación de la CLI de Greengrass durante la configuración inicial, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

Requisitos previos

Para implementar el componente de la CLI de Greengrass, asegúrese de que cumple los siguientes requisitos:

- AWS IoT Greengrass El software principal está instalado y configurado en el dispositivo principal. Para obtener más información, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

- Para utilizar el AWS CLI para implementar la CLI de Greengrass, debe haber instalado y configurado el. AWS CLI Para obtener más información, consulte Configuración de la AWS CLI en la Guía del usuario de AWS Command Line Interface .
- Debe estar autorizado a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Realice una de las siguientes acciones para usar la CLI de Greengrass:
  - Utilice el usuario del sistema que ejecuta el software AWS IoT Greengrass Core.
  - Utilice un usuario con permisos raíz o administrativos. En los dispositivos principales de Linux, puede utilizar `sudo` para obtener permisos de raíz.
  - Utilice un usuario del sistema que esté en un grupo que especifique en los parámetros de configuración `AuthorizedPosixGroups` o `AuthorizedWindowsGroups` al implementar el componente. Para obtener más información, consulte Configuración del componente de la CLI de Greengrass.

Implementación del componente de la CLI de Greengrass

Complete los siguientes pasos para implementar el componente de la CLI de Greengrass en su dispositivo principal:

Cómo implementar el componente de la CLI de Greengrass (consola)

1. Inicie sesión en la consola de AWS IoT Greengrass.
2. En el menú de navegación, elija Componentes.
3. En la página Componentes, en la pestaña Componentes públicos, elija `aws.greengrass.Cli`.
4. En la página aws.greengrass.Cli, elija Implementar.
5. En Agregar a la implementación, elija Crear nueva implementación.
6. En la página Especificar destino, en Objetivos de implementación, en la lista Nombre de destino, elija el grupo de Greengrass en el que desee realizar la implementación y elija Siguiente.
7. En la página Seleccionar componentes, compruebe que el componente aws.greengrass.Cli esté seleccionado y elija Siguiente.
8. En la página Configurar componentes, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
10. En la página Revisar, elija Implementar.

Cómo implementar el componente de la CLI de Greengrass (AWS CLI)

1. En su dispositivo, cree un archivo `deployment.json` que defina la configuración de implementación del componente de la CLI de Greengrass. Este archivo debería tener el siguiente aspecto:

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\":\"<group1>,<group2>,...,<groupN>\",
\"AuthorizedWindowsGroups\":\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

   - En el campo `target`, sustituya *`targetArn`* por el Nombre de recurso de Amazon (ARN) del objeto o grupo de objetos a la que apunte la implementación, en el siguiente formato:
     - Cosa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Grupo de cosas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
   - En el objeto del componente `aws.greengrass.Cli`, especifique los valores de la siguiente manera:

     `version`

     La versión del componente de la CLI de Greengrass.

     `configurationUpdate.AuthorizedPosixGroups`

     (Opcional) Una cadena que contiene una lista de grupos de sistema separados por comas. Usted autoriza a estos grupos de sistemas a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Puede especificar los nombres de los grupos o grupos IDs. Por ejemplo, `group1,1002,group3` autoriza a tres grupos de sistemas (`group1`, `1002` y `group3`) a utilizar la CLI de Greengrass.

     Si no especifica ningún grupo para autorizarlo, puede usar la CLI de Greengrass como usuario root (`sudo`) o como usuario del sistema que ejecuta el software AWS IoT Greengrass Core.

`configurationUpdate.AuthorizedWindowsGroups`

(Opcional) Una cadena que contiene una lista de grupos de sistema separados por comas. Usted autoriza a estos grupos de sistemas a utilizar la CLI de Greengrass para interactuar con el software AWS IoT Greengrass principal. Puede especificar los nombres de los grupos o grupos IDs. Por ejemplo, `group1,1002,group3` autoriza a tres grupos de sistemas (`group1`, `1002` y `group3`) a utilizar la CLI de Greengrass.

Si no especifica ningún grupo para autorizarlo, puede usar la CLI de Greengrass como administrador o como usuario del sistema que ejecuta el software AWS IoT Greengrass principal.

2. Ejecute el siguiente comando para implementar los componentes de la CLI de Greengrass en el dispositivo:

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/
to/deployment.json
```

Durante la instalación, el componente agrega un enlace simbólico a `greengrass-cli` en la carpeta */greengrass/v2*`/bin` del dispositivo y usted ejecuta la CLI de Greengrass desde esta ruta. Para ejecutar la CLI de Greengrass sin su ruta absoluta, agregue la carpeta */greengrass/v2*`/bin` a la variable PATH. Para verificar la instalación de la CLI de Greengrass, ejecute el comando siguiente:

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

Debería ver los siguientes datos de salida:

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

      --ggcRootPath=<ggcRootPath>
                  The AWS IoT Greengrass V2 root directory.
```

```
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  help                Show help information for a command.
  component           Retrieve component information and stop or restart
                        components.
  deployment          Create local deployments and retrieve deployment status.
  logs                Analyze Greengrass logs.
  get-debug-password  Generate a password for use with the HTTP debug view
                        component.
```

Si no encuentra `greengrass-cli`, es posible que la implementación no haya podido instalar la CLI de Greengrass. Para obtener más información, consulte Solución de problemas AWS IoT Greengrass V2.

Comandos de la CLI de Greengrass

La CLI de Greengrass proporciona una interfaz de línea de comandos para interactuar localmente con el dispositivo principal de AWS IoT Greengrass. Los comandos de la CLI de Greengrass utilizan el siguiente formato.

```
$ greengrass-cli <command> <subcommand> [arguments]
```

De forma predeterminada, el archivo ejecutable `greengrass-cli` de la carpeta */greengrass/v2*`/bin/` interactúa con la versión del software AWS IoT Greengrass Core que se ejecuta en la carpeta */greengrass/v2*. Si llama a un archivo ejecutable que no se encuentra en esta ubicación o si desea interactuar con el software AWS IoT Greengrass Core en una ubicación diferente, debe utilizar uno de los siguientes métodos para especificar de forma explícita la ruta raíz del software AWS IoT Greengrass Core con el que desea interactuar:

- Establezca la variable de entorno `GGC_ROOT_PATH` en */greengrass/v2*.
- Agregue los argumentos `--ggcRootPath` */greengrass/v2* a su comando como se muestra en el siguiente ejemplo.

  ```
  greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
  ```

Puede utilizar los siguientes argumentos con cualquier comando:

- Utilice `--help` para obtener información sobre un comando de la CLI específico de Greengrass.

- Utilice `--version` para obtener información sobre la versión de la CLI de Greengrass.

En esta sección, se describen los comandos de la CLI de Greengrass y se proporcionan ejemplos de estos comandos. La sinopsis de cada comando muestra sus argumentos y cómo se utilizan. Los argumentos opcionales deben ir entre corchetes.

Comandos disponibles

- componente
- implementación
- registros
- get-debug-password

componente

Utilice el comando `component` para interactuar con los componentes locales del dispositivo principal.

Subcomandos

- Detalles
- list
- reiniciar
- parar

Detalles

Recupera la versión, el estado y la configuración de un componente.

Sinopsis

```
greengrass-cli component details --name <component-name>
```

Argumentos

`--name`, `-n`. El nombre del componente.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ sudo greengrass-cli component details --name MyComponent

Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

list

Recupera el nombre, la versión, el estado y la configuración de cada componente instalado en el dispositivo.

Sinopsis

```
greengrass-cli component list
```

Argumentos

Ninguno

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ sudo greengrass-cli component list

Components currently running in Greengrass:
Component Name: FleetStatusService
Version: 0.0.0
State: RUNNING
Configuration: {"periodicUpdateIntervalSec":86400.0}
Component Name: UpdateSystemPolicyService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Nucleus
Version: 2.0.0
State: FINISHED
Configuration: {"awsRegion":"region","runWithDefault":
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}
Component Name: DeploymentService
Version: 0.0.0
```

```
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

reiniciar

Reinicie los componentes.

Sinopsis

```
greengrass-cli component restart --names <component-name>,...
```

Argumentos

`--names`, `-n`. El nombre del componente. Se requiere al menos el nombre de un componente. Puede especificar nombres de componentes adicionales, separando cada nombre con una coma.

Output

Ninguno

parar

Detenga la ejecución de los componentes.

Sinopsis

```
greengrass-cli component stop --names <component-name>,...
```

Argumentos

`--names`, `-n`. El nombre del componente. Se requiere al menos el nombre de un componente. Si es necesario, puede especificar nombres de componentes adicionales, separando cada nombre con una coma.

Output

Ninguno

implementación

Utilice el comando `deployment` para interactuar con los componentes locales del dispositivo principal.

Para supervisar el progreso de una implementación local, use el subcomando `status`. No puede supervisar el progreso de una implementación local mediante la consola.

Subcomandos

- crear
- cancelar
- list
- status

crear

Cree o actualice una implementación local mediante recetas de componentes, artefactos y argumentos de tiempo de ejecución específicos.

Sinopsis

```
greengrass-cli deployment create
    --recipeDir path/to/component/recipe
    [--artifactDir path/to/artifact/folder ]
    [--update-config {component-configuration}]
    [--groupId <thing-group>]
    [--merge "<component-name>=<component-version>"]...
    [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"]...
    [--systemLimits "{component-system-resource-limits}]"]...
    [--remove <component-name>,...]
    [--failure-handling-policy <policy name[ROLLBACK, DO_NOTHING]>]
```

Argumentos

- `--recipeDir`, `-r`. La ruta completa a la carpeta que contiene los archivos de recetas de los componentes.

- `--artifactDir`, `-a`. La ruta completa a la carpeta que contiene los archivos de artefactos que desea incluir en su implementación. La carpeta de artefactos debe contener la siguiente estructura de directorios:

  ```
  /path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
  ```

- `--update-config`, `-c`. Los argumentos de configuración de la implementación, proporcionados como una cadena JSON o un archivo JSON. La cadena JSON debe tener el siguiente formato:

  ```
  { \
    "componentName": { \
      "MERGE": {"config-key": "config-value"}, \
      "RESET": ["path/to/reset/"] \
    } \
  }
  ```

  `MERGE` y `RESET` distinguen entre mayúsculas y minúsculas y deben estar en mayúsculas.

- `--groupId`, `-g`. El grupo de objetos de destino de la implementación.
- `--merge`, `-m`. El nombre y la versión del componente de destino que desea agregar o actualizar. Debe proporcionar la información del componente en el formato *`<component>`*=*`<version>`*. Utilice un argumento diferente para cada componente adicional que desee especificar. Si es necesario, utilice el argumento `--runWith` para proporcionar información de `posixUser`, `posixGroup` y `windowsUser` para ejecutar el componente.
- `--runWith`. La información de `posixUser`, `posixGroup` y `windowsUser` para ejecutar un componente genérico o de Lambda. Debe proporcionar la información con el formato *`<component>`*`:{posixUser|windowsUser}=`*`<user>`*`[:`*`<=posixGroup>`*`]`. Por ejemplo, puede especificar **`HelloWorld:posixUser=ggc_user:ggc_group`** o **`HelloWorld:windowsUser=ggc_user`**. Utilice un argumento diferente para cada opción adicional que desee especificar.

  Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

- `--systemLimits`. Los límites de recursos del sistema que se aplicarán a los procesos de los componentes de Lambda genéricos y no contenerizados en el dispositivo principal. Puede configurar la cantidad máxima de uso de CPU y RAM que cada proceso de un componente

puede utilizar. Especifique un objeto JSON serializado o la ruta de un archivo a un archivo JSON. El objeto JSON debe incluir el siguiente formato.

```
{  \
  "componentName": { \
    "cpus": cpuTimeLimit, \
    "memory": memoryLimitInKb \
  } \
}
```

Puede configurar los siguientes límites de recursos del sistema para cada componente:

- `cpus`: la cantidad máxima de tiempo de CPU que los procesos de este componente pueden usar en el dispositivo principal. El tiempo total de CPU de un dispositivo principal equivale a la cantidad de núcleos de CPU del dispositivo. Por ejemplo, en un dispositivo principal con 4 núcleos de CPU, puede establecer este valor en 2 para limitar los procesos del componente al 50 % de uso de cada núcleo de CPU. En un dispositivo con 1 núcleo de CPU, puede establecer este valor en `0.25` para limitar los procesos del componente al 25 % de uso de la CPU. Si se establece este valor en un número mayor que la cantidad de núcleos de la CPU, el software AWS IoT Greengrass Core no limita el uso de la CPU del componente.
- `memory`: la cantidad máxima de RAM, expresada en kilobytes, que los procesos de un componente pueden usar en el dispositivo principal.

Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes.

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass y la CLI de Greengrass en los dispositivos principales de Linux. AWS IoT Greengrass actualmente no admite esta característica en los dispositivos principales de Windows.

- `--remove`. El nombre del componente de destino que desea eliminar de una implementación local. Para eliminar un componente que se combinó de una implementación en la nube, debe proporcionar el ID de grupo del grupo de objetos de destino en el siguiente formato:

  Greengrass nucleus v2.4.0 and later

  ```
  --remove <component-name> --groupId <group-name>
  ```

Earlier than v2.4.0

```
--remove <component-name> --groupId thinggroup/<group-name>
```

- `--failure-handling-policy`. Defina la acción que se lleva a cabo cuando una implementación da error. Hay dos acciones que puede especificar:
  - `ROLLBACK` –
  - `DO_NOTHING` –

  Esta característica está disponible para versión 2.11.0 y versiones posteriores de Núcleo de Greengrass.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ sudo greengrass-cli deployment create \
    --merge MyApp1=1.0.0 \
    --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \
    --remove MyApp3 \
    --recipeDir recipes/ \
    --artifactDir artifacts/

Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-
ad89-5151213dfcbc
```

cancelar

Cancela la implementación especificada.

Sinopsis

```
greengrass-cli deployment cancel
    -i <deployment-id>
```

Argumentos

`-i`. El identificador único de la implementación que se va a cancelar. El ID de implementación se devuelve en el resultado del comando `create`.

Output

- Ninguno

list

Recupere el estado de las últimas 10 implementaciones locales.

Sinopsis

```
greengrass-cli deployment list
```

Argumentos

Ninguno

Output

El siguiente ejemplo muestra los resultados del comando. Según el estado de la implementación, el resultado muestra uno de los siguientes valores de estado: `IN_PROGRESS`, `SUCCEEDED` o `FAILED`.

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

Recupere el estado de una implementación específica.

Sinopsis

```
greengrass-cli deployment status -i <deployment-id>
```

Argumentos

`-i`. El ID de la implementación.

Output

El siguiente ejemplo muestra los resultados del comando. Según el estado de la implementación, el resultado muestra uno de los siguientes valores de estado: `IN_PROGRESS`, `SUCCEEDED` o `FAILED`.

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

registros

Utilice el `logs` comando para analizar los registros de Greengrass en el dispositivo principal.

Subcomandos

- introducción
- list-keywords
- list-log-files

introducción

Recopile, filtre y visualice los archivos de registro de Greengrass. Este comando solo admite archivos de registro con formato JSON. Puede especificar el formato de registro en la configuración del núcleo.

Sinopsis

```
greengrass-cli logs get
    [--log-dir path/to/a/log/folder]
    [--log-file path/to/a/log/file]
    [--follow true | false ]
    [--filter <filter> ]
```

```
    [--time-window <start-time>,<end-time> ]
    [--verbose ]
    [--no-color ]
    [--before <value> ]
    [--after <value> ]
    [--syslog ]
    [--max-long-queue-size <value> ]
```

Argumentos

- `--log-dir`, `-ld`. La ruta al directorio para comprobar si hay archivos de registro, por ejemplo, ***/greengrass/v2*/logs**. No lo use con `--syslog`. Utilice un argumento diferente para cada directorio adicional que desee especificar. Debe especificar al menos una opción entre `--log-dir` o `--log-file`. También puede utilizar ambos argumentos en un único comando.
- `--log-file`, `-lf`. Las rutas de los directorios de registro que desee utilizar. Utilice un argumento diferente para cada directorio adicional que desee especificar. Debe especificar al menos una opción entre `--log-dir` o `--log-file`. También puede utilizar ambos argumentos en un único comando.
- `--follow`, `-fol`. Muestra las actualizaciones de registro a medida que se producen. La CLI de Greengrass continúa ejecutándose y lee los registros especificados. Si especifica un intervalo de tiempo, la CLI de Greengrass deja de inspeccionar los registros una vez finalizados todos los intervalos de tiempo.
- `--filter`, `-f`. La palabra clave, las expresiones regulares o el par clave-valor que se va a utilizar como filtro. Proporcione este valor como cadena, expresión regular o par clave-valor. Utilice un argumento diferente para cada filtro adicional que desee especificar.

  Cuando se evalúan, los filtros múltiples especificados en un único argumento se separan mediante operadores OR y los filtros especificados en argumentos adicionales se combinan con los operadores AND. Por ejemplo, si su comando incluye `--filter "installed" --filter "name=alpha,name=beta"`, la CLI de Greengrass filtrará y mostrará los mensajes de registro que contengan tanto la palabra clave `installed` como una clave `name` que incluya los valores `alpha` o `beta`.
- `--time-window`, `-t`. El intervalo de tiempo durante el que se mostrará la información del registro. Puede utilizar tanto marcas temporales exactas como compensaciones relativas. Debe proporcionar la información con el formato *<begin-time>*,*<end-time>*. Si no especifica la hora de inicio ni la hora de finalización, el valor de esa opción se establece de forma predeterminada en la fecha y hora actuales del sistema. Utilice un argumento diferente para cada intervalo de tiempo adicional que desee especificar.

La CLI de Greengrass admite los siguientes formatos para las marcas temporales:

- `yyyy-MM-DD`, por ejemplo, `2020-06-30`. La hora predeterminada es 00:00:00 cuando se utiliza este formato.

  `yyyyMMDD`, por ejemplo, `20200630`. La hora predeterminada es 00:00:00 cuando se utiliza este formato.

  `HH:mm:ss`, por ejemplo, `15:30:45`. La fecha por defecto es la fecha actual del sistema cuando se utiliza este formato.

  `HH:mm:ssSSS`, por ejemplo, `15:30:45`. La fecha establece de forma predeterminada la fecha actual del sistema cuando se utiliza este formato.

  `YYYY-MM-DD'T'HH:mm:ss'Z'`, por ejemplo, `2020-06-30T15:30:45Z`.

  `YYYY-MM-DD'T'HH:mm:ss`, por ejemplo, `2020-06-30T15:30:45`.

  `yyyy-MM-dd'T'HH:mm:ss.SSS`, por ejemplo, `2020-06-30T15:30:45.250`.

  Las compensaciones relativas especifican un desfase del periodo con respecto a la hora actual del sistema. La CLI de Greengrass admite el siguiente formato para las compensaciones relativas: `+|-[<value>h|hr|hours][value m|min|minutes][value]s|sec|seconds`.

  Por ejemplo, el siguiente argumento para especificar un intervalo de tiempo entre 1 hora y 2 horas y 15 minutos antes de la hora actual es `--time-window -2h15min,-1hr`.
- `--verbose`. Muestre todos los campos de los mensajes de registro. No lo use con `--syslog`.
- `--no-color`, `-nc`. Elimine el código de colores. El código de colores predeterminado para los mensajes de registro utiliza texto rojo en negrita. Solo admite terminales tipo UNIX porque utiliza secuencias de escape ANSI.
- `--before`, `-b`. El número de líneas que se deben mostrar antes de una entrada de registro coincidente. El valor predeterminado es 0.
- `--after`, `-a`. El número de líneas que se muestran después de una entrada de registro coincidente. El valor predeterminado es 0.
- `--syslog`. Procese todos los archivos de registro mediante el protocolo syslog definido en el RFC3164. No lo use con `--log-dir` y `--verbose`. El protocolo syslog usa el siguiente formato: `"<$Priority>$Timestamp $Host $Logger ($Class): $Message"`. Si no especifica un archivo de registro, la CLI de Greengrass lee los mensajes de registro

de las siguientes ubicaciones: `/var/log/messages`, `/var/log/syslog` o `/var/log/system.log`.

AWS IoT Greengrass actualmente no admite esta característica en los dispositivos principales de Windows.

- `--max-log-queue-size`, `-m`. El número máximo de entradas de registro que se van a asignar a la memoria. Utilice esta opción para optimizar el uso de la memoria. El valor predeterminado es 100.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ sudo greengrass-cli logs get --verbose \
    --log-file /greengrass/v2/logs/greengrass.log \
    --filter deployment,serviceName=DeploymentService \
    --filter level=INFO \
    --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22

2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.DeploymentService: Current deployment finished.
 {DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,
 currentState=RUNNING}
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted
 deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

list-keywords

Muestra las palabras clave sugeridas que puede usar para filtrar los archivos de registro.

Sinopsis

```
greengrass-cli logs list-keywords [arguments]
```

Argumentos

Ninguno

Output

Los siguientes ejemplos muestran los resultados del comando.

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog

Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

Muestra los archivos de registro que se encuentran en el directorio especificado.

Sinopsis

```
greengrass-cli logs list-log-files [arguments]
```

Argumentos

`--log-dir`, `-ld`. La ruta al directorio para comprobar los archivos de registro.

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
/greengrass/v2/logs/main.log
```

```
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

Utilice el comando `get-debug-password` para imprimir una contraseña generada aleatoriamente para el componente de la consola de depuración local (`aws.greengrass.LocalDebugConsole`). La contraseña caduca 8 horas después de generarse.

Sinopsis

```
greengrass-cli get-debug-password
```

Argumentos

Ninguno

Output

El siguiente ejemplo muestra los resultados del comando.

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
 console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

Utilice el marco AWS IoT Greengrass de pruebas

Greengrass Testing Framework (GTF) es un conjunto de componentes básicos que respaldan la end-to-end automatización desde la perspectiva del cliente. GTF utiliza Cucumber como motor de funciones. AWS IoT Greengrass utiliza los mismos componentes básicos para calificar los cambios

de software en varios dispositivos. Para obtener más información, consulte Greengrass Testing Framework en Github.

El GTF se implementa con Cucumber, una herramienta que se utiliza para ejecutar pruebas automatizadas, a fin de fomentar un desarrollo impulsado por el comportamiento (BDD) de los componentes. En Cucumber, las características de este sistema se describen en un tipo especial de archivo llamado `feature`. Cada característica se describe en un formato legible por humanos llamado escenarios, que son especificaciones que se pueden convertir en pruebas automatizadas. Cada escenario se describe como una serie de pasos que definen las interacciones y los resultados del sistema que se está probando con un lenguaje de dominio específico llamado Gherkin. Un paso de Gherkin se vincula al código de programación mediante un método llamado definición de pasos que conecta la especificación al flujo de prueba. Las definiciones de pasos en GTF se implementan con Java.

Temas

- Funcionamiento
- Registros de cambios
- Opciones de configuración de Greengrass Testing Framework
- Tutorial: Ejecute end-to-end pruebas con Greengrass Testing Framework y Greengrass Development Kit
- Tutorial: Uso de una prueba de confianza del conjunto de pruebas de confianza

Funcionamiento

AWS IoT Greengrass distribuye el GTF como un JAR independiente que consta de varios módulos de Java. Para utilizar el GTF para end-to-end probar componentes, debe implementar las pruebas en un proyecto de Java. Agregar el JAR compatible con las pruebas como dependencia en el proyecto de Java le permite utilizar la funcionalidad existente del GTF y ampliarla mediante la escritura de sus propios casos de prueba personalizados. Para ejecutar los casos de prueba personalizados, puede crear el proyecto Java y ejecutar el JAR de destino con las opciones de configuración que se describen en Opciones de configuración de Greengrass Testing Framework.

JAR independiente del GTF

Greengrass utiliza Cloudfront como repositorio de Maven para alojar diferentes versiones del JAR independiente del GTF. Para obtener una lista completa de las versiones del GTF, consulte los lanzamientos de GTF.

El JAR independiente del GTF incluye los siguientes módulos. No se limita solo a estos módulos. Puede seleccionar cada una de estas dependencias por separado en su proyecto o incluirlas todas a la vez en el archivo JAR independiente de prueba.

- `aws-greengrass-testing-resources`: Este módulo proporciona una abstracción para gestionar el ciclo de vida de un AWS recurso durante el transcurso de una prueba. Puedes usarlo para definir tus AWS recursos personalizados mediante la `ResourceSpec` abstracción, de modo que GTF pueda encargarse de crear y eliminar esos recursos por ti.
- `aws-greengrass-testing-platform`: este módulo proporciona una abstracción en la plataforma para el dispositivo que se prueba durante el ciclo de vida de la prueba. Contiene una herramienta APIs para interactuar con el sistema operativo independientemente de la plataforma y se puede utilizar para simular los comandos que se ejecutan en la consola del dispositivo.
- `aws-greengrass-testing-components`: este módulo consta de componentes de muestra que se utilizan para probar las características principales de Greengrass, como las implementaciones, el IPC y otras características.
- `aws-greengrass-testing-features`: este módulo consta de pasos comunes reutilizables y las definiciones que se utilizan para realizar pruebas en el entorno de Greengrass.

Temas

- Registros de cambios
- Opciones de configuración de Greengrass Testing Framework
- Tutorial: Ejecute end-to-end pruebas con Greengrass Testing Framework y Greengrass Development Kit
- Tutorial: Uso de una prueba de confianza del conjunto de pruebas de confianza

Registros de cambios

En la siguiente tabla, se describen los cambios en cada versión del GTF. Para obtener más información, consulte la página de versiones de GTF en GitHub.

Versión	Cambios
1.2.0	Nuevas características • Agrega pasos relacionados con la red para configurar el MQTT y la conectividad a la red de Internet durante las pruebas.

Versión	Cambios
	• Agrega pasos métricos del sistema para monitorear el uso de la RAM y la CPU del dispositivo. Mejoras y correcciones de errores • El paso de implementación local de la CLI de Greengrass se vuelve a intentar hasta que se realiza correctamente. • Las pruebas detienen con cautela el núcleo de Greengrass, en lugar de hacerlo directamente. • Añade una mejora en la que GTF sondea el punto final de AWS IoT credenciales hasta que se puedan recuperar las credenciales del alias de la cosa y del rol. • Corrige los artefactos y directorios de recetas que faltaban. Esta versión también corrige las versiones de los componentes que faltan. • Soluciona un problema por el que GTF fallaba durante la limpieza de la imagen de docker si la imagen de docker no existe. • Agrega la palabra clave CURRENT como versión del componente.
1.1.0	Nuevas características • Agrega la posibilidad de instalar un componente personalizado con la configuración. Esto requiere una receta para el componente personali zado. • Agrega la posibilidad de actualizar una implementación local con una configuración personalizada. Mejoras y correcciones de errores • Corrige el problema de inconsistencia de la versión de GTF en el contexto del registro.
1.0.0	Versión inicial.

Opciones de configuración de Greengrass Testing Framework

Opciones de configuración de GTF

Greengrass Testing Framework (GTF) le permite configurar ciertos parámetros durante el inicio del proceso de prueba integral para orquestrar el flujo de pruebas. Puede especificar estas opciones de configuración como argumentos de la CLI para el JAR independiente de GTF.

La versión 1.1.0 y posteriores de GTF ofrecen las siguientes opciones de configuración.

- `additional-plugins`: (opcional) complementos de Cucumber adicionales
- `aws-region`: apunta a puntos de conexión regionales específicos para los servicios de AWS. El valor predeterminado es lo que detecta el SDK de AWS.
- `credentials-path`: ruta de credenciales de perfil de AWS opcional. El valor predeterminado son las credenciales detectadas en el entorno host.
- `credentials-path-rotation`: duración de rotación opcional para las credenciales de AWS. El valor predeterminado es de 15 minutos o `PT15M`.
- `csr-path`: la ruta de la CSR mediante la cual se generará el certificado del dispositivo.
- `device-mode`: el dispositivo objetivo que se está probando. El valor predeterminado es el dispositivo local.
- `env-stage`: apunta al entorno de implementación de Greengrass. El valor predeterminado es de producción.
- `existing-device-cert-arn`: el ARN de un certificado existente que desee usar como certificado de dispositivo para Greengrass.
- `feature-path`: archivo o directorio que contiene archivos de características adicionales. El valor predeterminado es que no se usan archivos de características adicionales.
- `gg-cli-version`: anula la versión de la CLI de Greengrass. El valor predeterminado es el que se encuentra en `ggc.version`.
- `gg-component-bucket`: el nombre de un bucket de Amazon S3 existente que aloja los componentes de Greengrass.
- `gg-component-overrides`: una lista de anulaciones de componentes de Greengrass.
- `gg-persist`: una lista de elementos de prueba que persisten tras una ejecución de la prueba. El comportamiento predeterminado es no conservar nada. Los valores aceptados son: `aws.resources`, `installed.software` y `generated.files`.

- `gg-runtime`: una lista de valores para influir en la forma en que la prueba interactúa con los recursos de la prueba. Estos valores sustituyen al parámetro `gg.persist`. Si el valor predeterminado está vacío, se asume que todos los recursos de prueba se administran por caso de prueba, incluido el tiempo de ejecución de Greengrass instalado. Los valores aceptados son: `aws.resources`, `installed.software` y `generated.files`.
- `ggc-archive`: la ruta hacia el componente del núcleo de Greengrass archivado.
- `ggc-install-root`: directorio para instalar el componente núcleo de Greengrass. Los valores predeterminados son test.temp.path y la carpeta de ejecución de la prueba.
- `ggc-log-level`: establece el nivel de registro del núcleo de Greengrass para la ejecución de la prueba. El valor predeterminado es “INFO”.
- `ggc-tes-rolename`: el rol de IAM que asumirá AWS IoT Greengrass Core para acceder a los servicios de AWS. Si no existe un rol con un nombre dado, se creará uno con una política de acceso predeterminada.
- `ggc-trusted-plugins`: la lista separada por comas de las rutas (en el host) de los complementos de confianza que deben agregarse a Greengrass. Para indicar la ruta en el propio DUT, agregue el prefijo “dut:” a la ruta
- `ggc-user-name`: el valor user:group posixUser para el núcleo de Greengrass. El valor predeterminado es el nombre de usuario actual con el que se ha iniciado sesión.
- `ggc-version`: anula la versión del componente núcleo de Greengrass en ejecución. El valor predeterminado es el que se encuentra en ggc.archive.
- `log-level`: nivel de registro de la ejecución de la prueba. El valor predeterminado es “INFO”.
- `parallel-config`: conjunto de índices de lotes y número de lotes como cadena JSON. El valor predeterminado del índice de lotes es 0 y el número de lotes es 1.
- `proxy-url`: configura todas las pruebas para enrutar el tráfico a través de esta URL.
- `tags`: ejecuta únicamente etiquetas de características. Se puede intersecar con “&”
- `test-id-prefix`: un prefijo común que se aplica a todos los recursos específicos de la prueba, incluidos los nombres y las etiquetas de los recursos de AWS. El prefijo predeterminado es “gg”.
- `test-log-path`: directorio que contendrá los resultados de toda la ejecución de la prueba. El valor predeterminado es “testResults”.
- `test-results-json`: marcador para determinar si se genera un informe JSON de Cucumber resultante escrito en el disco. El valor predeterminado es true (verdadero).
- `test-results-log`: marcador para determinar si la salida de la consola se genera escrita en el disco. El valor predeterminado es falso.

- `test-results-xml`: marcador para determinar si se genera un informe XML de JUnit resultante escrito en el disco. El valor predeterminado es true (verdadero).
- `test-temp-path`: directorio para generar artefactos de prueba locales. El valor predeterminado es un directorio temporal aleatorio con el prefijo gg-testing.
- `timeout-multiplier`: multiplicador proporcionado para todos los tiempos de espera de las pruebas. El valor predeterminado es 1.0.

Tutorial: Ejecute end-to-end pruebas con Greengrass Testing Framework y Greengrass Development Kit

AWS IoT Greengrass Testing Framework (GTF) y Greengrass Development Kit (GDK) ofrecen a los desarrolladores formas de ejecutar pruebas. end-to-end Puede completar este tutorial para inicializar un proyecto de GDK con un componente, inicializar un proyecto de GDK con un módulo de prueba y crear un caso de end-to-end prueba personalizado. Una vez que haya compilado su caso de prueba personalizado, podrá ejecutar la prueba.

En este tutorial, aprenderá a hacer lo siguiente:

1. Inicialice un proyecto de GDK con un componente.
2. Inicialice un proyecto de GDK con un módulo de prueba. end-to-end
3. Cree un caso de prueba personalizado.
4. Agregue una etiqueta al nuevo caso de prueba.
5. Compile el JAR de prueba.
6. Ejecute la prueba .

Temas

- Requisitos previos
- Paso 1: Inicializar un proyecto de GDK con un componente
- Paso 2: inicializar un proyecto de GDK con un módulo de prueba end-to-end
- Paso 3: Crear un caso de prueba personalizado
- Paso 4: Agregar una etiqueta al nuevo caso de prueba
- Paso 5: Compilar el JAR de prueba
- Paso 6: Ejecutar la prueba
- Ejemplo: crear un caso de prueba personalizado

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- GDK versión 1.3.0 o posterior
- Java
- Maven
- Git

Paso 1: Inicializar un proyecto de GDK con un componente

- Inicialice una carpeta vacía con un proyecto de GDK. Para descargar el componente `HelloWorld` implementado en Python, ejecute el siguiente comando.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  Este comando crea un directorio denominado `HelloWorld` en el directorio actual.

Paso 2: inicializar un proyecto de GDK con un módulo de prueba end-to-end

- GDK le permite descargar la plantilla del módulo de pruebas que consta de una característica y una implementación escalonada. Ejecute el siguiente comando para abrir el directorio `HelloWorld` e inicializar el proyecto GDK existente mediante un módulo de pruebas.

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  Este comando crea un directorio denominado `gg-e2e-tests` dentro del directorio `HelloWorld`. Este directorio de pruebas es un proyecto de Maven que depende del JAR independiente de pruebas de Greengrass.

Paso 3: Crear un caso de prueba personalizado

La redacción de un caso de prueba personalizado consta, en líneas generales, de dos pasos: crear un archivo de características con un escenario de prueba e implementar las definiciones de los pasos. Para ver un ejemplo de cómo crear un caso de prueba personalizado, consulte Ejemplo: crear

un caso de prueba personalizado. Siga los pasos que se describen a continuación para crear un caso de prueba personalizado:

1. Cree un archivo de características con un escenario de prueba

   Por lo general, una característica describe una funcionalidad específica del software que se está probando. En Cucumber, cada característica se especifica como un archivo de características individual con un título, una descripción detallada y uno o más ejemplos de casos específicos denominados escenarios. Cada escenario consta de un título, una descripción detallada y una serie de pasos que definen las interacciones y los resultados esperados. Los escenarios se escriben en un formato estructurado con las palabras clave “dado”, “cuándo” y “entonces”.

2. Implementación de definiciones de pasos

   Una definición de paso vincula el paso de Gherkin en lenguaje sencillo con el código programático. Cuando Cucumber identifique un paso de Gherkin en un escenario, buscará una definición de paso coincidente para ejecutarlo.

Paso 4: Agregar una etiqueta al nuevo caso de prueba

- Puede asignar etiquetas a las características y escenarios para organizar el proceso de prueba. Puede usar etiquetas para categorizar los subconjuntos de escenarios y también seleccionar los enlaces de forma condicional para que se ejecuten. Las características y los escenarios pueden tener varias etiquetas separadas por un espacio.

  En este ejemplo, estamos usando el componente `HelloWorld`.

  En el archivo de características, agrega una nueva etiqueta con un nombre `@HelloWorld` junto a la etiqueta `@Sample`.

  ```
  @Sample @HelloWorld
  Scenario: As a developer, I can create a component and deploy it on my device
  ....
  ```

Paso 5: Compilar el JAR de prueba

1. Compile el componente. Debe compilar el componente antes de crear el módulo de prueba.

   ```
   gdk component build
   ```

2. Compile el módulo de prueba con el siguiente comando. Este comando compilará el JAR de prueba en la carpeta `greengrass-build`.

```
gdk test-e2e build
```

Paso 6: Ejecutar la prueba

Cuando realiza una prueba de confianza, el GTF automatiza el ciclo de vida de la prueba y administra los recursos que se crearon durante la prueba. Primero aprovisiona un dispositivo bajo prueba (DUT) como una AWS IoT cosa e instala el software principal de Greengrass en él. A continuación, creará un nuevo componente denominado `HelloWorld` con la receta especificada en esa ruta. A continuación, el componente `HelloWorld` se implementa en el dispositivo principal mediante una implementación de objeto de Greengrass. A continuación, se verificará si la implementación se realiza correctamente. El estado de la implementación cambiará a `COMPLETED` dentro de 3 minutos si la implementación se realiza correctamente.

1. Vaya al archivo `gdk-config.json` en el directorio del proyecto para apuntar a las pruebas con la etiqueta `HelloWorld`. Actualice la clave `test-e2e` mediante el siguiente comando.

```
"test-e2e":{
  "gtf_options" : {
       "tags":"HelloWorld"
   }
}
```

2. Antes de ejecutar las pruebas, debe proporcionar AWS las credenciales al dispositivo anfitrión. El GTF usa estas credenciales para administrar los AWS recursos durante el proceso de prueba. Asegúrese de que el rol que proporcione tenga los permisos para automatizar las operaciones necesarias que se incluyen en la prueba.

   Ejecute los siguientes comandos para proporcionar las AWS credenciales.

   - Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

3. Ejecute la prueba con el siguiente comando.

```
gdk test-e2e run
```

Este comando descarga la última versión del núcleo de Greengrass de la carpeta `greengrass-build` y ejecuta pruebas con ella. Este comando también se dirige solo a los escenarios con la etiqueta `HelloWorld` y genera un informe para esos escenarios. Verá que los AWS recursos que se crearon durante esta prueba se descartan al final de la prueba.

Ejemplo: crear un caso de prueba personalizado

Example

El módulo de pruebas descargado en el proyecto GDK consta de un archivo de características proporcionado.

En el siguiente ejemplo, utilizamos un archivo de características para probar la característica de implementación del objeto del software Greengrass. Probamos parcialmente la funcionalidad de esta característica con un escenario que realiza la implementación de un componente a través de la Nube de AWS de Greengrass. Se trata de una serie de pasos que nos ayudan a entender las interacciones y los resultados esperados de este caso de uso.

1. Creación de un archivo de características

   Navegue hasta la carpeta `gg-e2e-tests/src/main/resources/greengrass/features` del directorio actual. Puede encontrar el ejemplo `component.feature` con el aspecto del siguiente ejemplo.

En este archivo de características, puede probar la característica de implementación de objetos del software Greengrass. Puede probar parcialmente la funcionalidad de esta característica con un escenario que realice la implementación de un componente a través de la nube de Greengrass. El escenario consiste en una serie de pasos que ayudan a comprender las interacciones y los resultados esperados de cada caso de prueba.

```
Feature: Testing features of Greengrassv2 component

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass

@Sample
Scenario: As a developer, I can create a component and deploy it on my device
    When I create a Greengrass deployment with components
        HelloWorld | /path/to/recipe/file
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    And I call my custom step
```

El GTF contiene las definiciones de todos los pasos siguientes, excepto el paso denominado: `And I call my custom step`.

2. Implementación de definiciones de pasos

   El JAR independiente de GTF contiene las definiciones de todos los pasos excepto uno: `And I call my custom step`. Puede implementar este paso en el módulo de pruebas.

   Navegue hasta el código de origen del archivo de prueba. Puede vincular su paso personalizado mediante una definición de paso utilizando el siguiente comando.

```
@And("I call my custom step")
public void customStep() {
    System.out.println("My custom step was called ");
}
```

Tutorial: Uso de una prueba de confianza del conjunto de pruebas de confianza

AWS IoT Greengrass Testing Framework (GTF) y Greengrass Development Kit (GDK) ofrecen a los desarrolladores formas de ejecutar pruebas. end-to-end Puede completar este tutorial para inicializar un proyecto de GDK con un componente, inicializar un proyecto de GDK con un módulo de prueba y utilizar una end-to-end prueba de confianza del conjunto de pruebas de confianza. Una vez que haya compilado su caso de prueba personalizado, podrá ejecutar la prueba.

Una prueba de confianza es una prueba genérica proporcionada por Greengrass que valida los comportamientos de los componentes fundamentales. Estas pruebas se pueden modificar o ampliar para adaptarlas a necesidades de componentes más específicas.

Para este tutorial utilizaremos un componente. HelloWorld Si está utilizando otro componente, sustituya el HelloWorld componente por el suyo.

En este tutorial, aprenderá a hacer lo siguiente:

1. Inicialice un proyecto de GDK con un componente.
2. Inicialice un proyecto de GDK con un módulo de end-to-end prueba.
3. Use una prueba del conjunto de pruebas de confianza.
4. Agregue una etiqueta al nuevo caso de prueba.
5. Compile el JAR de prueba.
6. Ejecute la prueba .

Temas

- Requisitos previos
- Paso 1: Inicializar un proyecto de GDK con un componente
- Paso 2: inicializar un proyecto de GDK con un módulo de prueba end-to-end
- Paso 3: Usar una prueba del conjunto de pruebas de confianza
- Paso 4: Agregar una etiqueta al nuevo caso de prueba
- Paso 5: Compilar el JAR de prueba
- Paso 6: Ejecutar la prueba
- Ejemplo: uso de una prueba de confianza

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- GDK versión 1.6.0 o posterior
- Java
- Maven
- Git

Paso 1: Inicializar un proyecto de GDK con un componente

- Inicialice una carpeta vacía con un proyecto de GDK. Para descargar el componente `HelloWorld` implementado en Python, ejecute el siguiente comando.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  Este comando crea un directorio denominado `HelloWorld` en el directorio actual.

Paso 2: inicializar un proyecto de GDK con un módulo de prueba end-to-end

- GDK le permite descargar la plantilla del módulo de pruebas que consta de una característica y una implementación escalonada. Ejecute el siguiente comando para abrir el directorio `HelloWorld` e inicializar el proyecto GDK existente mediante un módulo de pruebas.

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  Este comando crea un directorio denominado `gg-e2e-tests` dentro del directorio `HelloWorld`. Este directorio de pruebas es un proyecto de Maven que depende del JAR independiente de pruebas de Greengrass.

Paso 3: Usar una prueba del conjunto de pruebas de confianza

La redacción de un caso de prueba de confianza consiste en usar el archivo de características proporcionado y, si es necesario, modificar los escenarios. Para ver un ejemplo de uso de una prueba de confianza, consulte Ejemplo: crear un caso de prueba personalizado. Siga los pasos a continuación para usar una prueba de confianza:

- Use el archivo de características proporcionado.

  Navegue hasta la carpeta `gg-e2e-tests/src/main/resources/greengrass/features` del directorio actual. Abra el archivo de muestra `confidenceTest.feature` para usar la prueba de confianza.

Paso 4: Agregar una etiqueta al nuevo caso de prueba

- Puede asignar etiquetas a las características y escenarios para organizar el proceso de prueba. Puede usar etiquetas para categorizar los subconjuntos de escenarios y también seleccionar los enlaces de forma condicional para que se ejecuten. Las características y los escenarios pueden tener varias etiquetas separadas por un espacio.

  En este ejemplo, estamos usando el componente `HelloWorld`.

  Cada escenario está etiquetado con `@ConfidenceTest`. Cambie o agregue etiquetas si desea ejecutar solo un subconjunto del conjunto de pruebas. Cada escenario de prueba se describe en la parte superior de cada prueba de confianza. El escenario consiste en una serie de pasos que ayudan a comprender las interacciones y los resultados esperados de cada caso de prueba. Puede ampliar estas pruebas al agregar sus propios pasos o modificar los existentes.

  ```
  @ConfidenceTest
  Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
   is working as expected
  ....
  ```

Paso 5: Compilar el JAR de prueba

1. Compile el componente. Debe compilar el componente antes de crear el módulo de prueba.

   ```
   gdk component build
   ```

2. Compile el módulo de prueba con el siguiente comando. Este comando compilará el JAR de prueba en la carpeta `greengrass-build`.

   ```
   gdk test-e2e build
   ```

Paso 6: Ejecutar la prueba

Cuando realiza una prueba de confianza, el GTF automatiza el ciclo de vida de la prueba y administra los recursos que se crearon durante la prueba. Primero aprovisiona un dispositivo bajo prueba (DUT) como una AWS IoT cosa e instala el software principal de Greengrass en él. A continuación, creará un nuevo componente denominado `HelloWorld` con la receta especificada en esa ruta. A continuación, el componente `HelloWorld` se implementa en el dispositivo principal mediante una implementación de objeto de Greengrass. A continuación, se verificará si la implementación se realiza correctamente. El estado de la implementación cambiará a `COMPLETED` dentro de 3 minutos si la implementación se realiza correctamente.

1. Vaya al archivo `gdk-config.json` del directorio del proyecto para seleccionar las pruebas con la etiqueta `ConfidenceTest` o cualquier etiqueta que haya especificado en el Paso 4. Actualice la clave `test-e2e` mediante el siguiente comando.

   ```
   "test-e2e":{
     "gtf_options" : {
          "tags":"ConfidenceTest"
      }
   }
   ```

2. Antes de ejecutar las pruebas, debe proporcionar AWS las credenciales al dispositivo anfitrión. El GTF usa estas credenciales para administrar los AWS recursos durante el proceso de prueba. Asegúrese de que el rol que proporcione tenga los permisos para automatizar las operaciones necesarias que se incluyen en la prueba.

   Ejecute los siguientes comandos para proporcionar las AWS credenciales.

   - Linux or Unix

     ```
     export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

3. Ejecute la prueba con el siguiente comando.

```
gdk test-e2e run
```

Este comando descarga la última versión del núcleo de Greengrass de la carpeta `greengrass-build` y ejecuta pruebas con ella. Este comando también se dirige solo a los escenarios con la etiqueta `ConfidenceTest` y genera un informe para esos escenarios. Verá que los AWS recursos que se crearon durante esta prueba se descartan al final de la prueba.

Ejemplo: uso de una prueba de confianza

Example

El módulo de pruebas descargado en el proyecto del GDK consta de un archivo de características proporcionado.

En el siguiente ejemplo, usamos un archivo de características para probar la característica de implementación del objeto del software Greengrass. Probamos parcialmente la funcionalidad de esta característica con un escenario que realiza la implementación de un componente a través de la Nube de AWS de Greengrass. Se trata de una serie de pasos que nos ayudan a entender las interacciones y los resultados esperados de este caso de uso.

- Use el archivo de características proporcionado.

  Navegue hasta la carpeta `gg-e2e-tests/src/main/resources/greengrass/features` del directorio actual. Puede encontrar el ejemplo `confidenceTest.feature` con el aspecto del siguiente ejemplo.

```
Feature: Confidence Test Suite

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass
```

```
@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
    When I create a Greengrass deployment with components
      | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
      | aws.greengrass.Cli | LATEST                    |
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    # Update component state accordingly. Possible states: {RUNNING, FINISHED,
 BROKEN, STOPPING}
    And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-
cli
```

Cada escenario de prueba se describe en la parte superior de cada prueba de confianza. El escenario consiste en una serie de pasos que ayudan a comprender las interacciones y los resultados esperados de cada caso de prueba. Puede ampliar estas pruebas al agregar sus propios pasos o modificar los existentes. Cada uno de los escenarios incluye comentarios que lo ayudan a realizar estos ajustes.

Desarrollo de componentes de AWS IoT Greengrass

Puede desarrollar y probar componentes en su dispositivo principal de Greengrass. Como resultado, puede crear e iterar su software de AWS IoT Greengrass sin interactuar con la Nube de AWS. Cuando termine una versión de su componente, podrá subirla a AWS IoT Greengrass en la nube para que usted y su equipo puedan implementar el componente en otros dispositivos de su flota. Para obtener más información acerca de cómo implementar componentes, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Cada componente incluye una receta y artefactos.

- Recetas

  Cada componente contiene un archivo de recetas que define sus metadatos. La receta también especifica los parámetros de configuración, dependencias del componente, ciclo de vida y compatibilidad de plataforma del componente. El ciclo de vida del componente define los comandos que instalan, ejecutan y apagan el componente. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.

  Puede definir recetas en formato JSON o YAML.

- Artefactos

  Los componentes pueden tener cualquier número de artefactos, que son componentes binarios. Los artefactos pueden incluir scripts, código compilado, recursos estáticos y cualquier otro archivo que utilice un componente. Los componentes también pueden utilizar artefactos de las dependencias de los componentes.

AWS IoT Greengrass proporciona componentes prediseñados que puede utilizar en sus aplicaciones e implementar en sus dispositivos. Por ejemplo, puede usar el componente administrador de flujos para cargar datos en varios servicios de AWS o puede usar el componente de métricas de CloudWatch para publicar métricas personalizadas en Amazon CloudWatch. Para obtener más información, consulte Componentes proporcionados por AWS.

AWS IoT Greengrass prepara un índice de componentes de Greengrass, llamado Catálogo de software de Greengrass. Este catálogo rastrea los componentes de Greengrass desarrollados por la comunidad de Greengrass. Desde este catálogo, puede descargar, modificar e implementar componentes para crear sus aplicaciones de Greengrass. Para obtener más información, consulte Componentes de la comunidad.

El software AWS IoT Greengrass Core ejecuta los componentes como usuario y grupo del sistema, como `ggc_user` y `ggc_group`, que usted configura en el dispositivo principal. Esto significa que los componentes tienen los permisos de ese usuario del sistema. Si utiliza un usuario del sistema sin un directorio principal, los componentes no pueden utilizar comandos de ejecución ni código que utilice un directorio principal. Esto significa que no puede usar el comando `pip install some-library --user` para instalar paquetes de Python, por ejemplo. Si ha seguido el tutorial de introducción para configurar su dispositivo principal, el usuario del sistema no tiene un directorio principal. Para obtener más información sobre cómo configurar el usuario y el grupo que ejecutan los componentes, consulte Configuración del usuario que ejecuta los componentes.

> **Note**
>
> AWS IoT Greengrass usa versiones semánticas para los componentes. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. Por ejemplo, la versión `1.0.0` representa el primer lanzamiento principal de un componente. Para obtener más información, consulte la especificación semántica de la versión.

Temas

- Componente del ciclo de vida
- Tipos de componentes
- Creación de componentes de AWS IoT Greengrass
- Prueba de los componentes de AWS IoT Greengrass con implementaciones locales
- Publique componentes para desplegarlos en sus dispositivos principales
- Interacción con servicios de AWS
- Ejecución de un contenedor de Docker
- AWS IoT Greengrass referencia de recetas de componentes
- Referencia de variable de entorno del componente

Componente del ciclo de vida

El componente del ciclo de vida define las etapas que el software AWS IoT Greengrass Core utiliza para instalar y ejecutar los componentes. Cada etapa define un script y otra información que especifica cómo se comporta el componente. Por ejemplo, al instalar un componente, el software AWS IoT Greengrass Core ejecuta el script del ciclo de vida `install` de ese componente. Los componentes de los dispositivos principales tienen los siguientes estados de ciclo de vida:

- `NEW`: la receta y los artefactos del componente se cargan en el dispositivo principal, pero el componente no está instalado. Cuando un componente entra en este estado, ejecuta su script de instalación.
- `INSTALLED`: el componente está instalado en el dispositivo principal. El componente entra en este estado después de ejecutar su script de instalación.
- `STARTING`: el componente se está iniciando en el dispositivo principal. El componente entra en este estado cuando ejecuta su script de arranque. Si el arranque se realiza correctamente, el componente entra en el estado `RUNNING`.
- `RUNNING`: el componente se está ejecutando en el dispositivo principal. El componente entra en este estado cuando ejecuta su script de ejecución o cuando tiene procesos en segundo plano activos desde su script de arranque.
- `FINISHED`: el componente se ejecutó correctamente y completó su ejecución.
- `STOPPING`: el componente se detiene. El componente entra en este estado cuando ejecuta su script de apagado.
- `ERRORED`: el componente ha detectado un error. Cuando el componente entra en este estado, ejecuta su script de recuperación. A continuación, el componente se reinicia para intentar

volver a su uso normal. Si el componente entra en el estado `ERRORED` tres veces sin ejecutarse correctamente, pasa a ser `BROKEN`.

- `BROKEN`: el componente ha detectado errores varias veces y no se puede recuperar. Debe volver a implementar el componente para solucionarlo.

Tipos de componentes

El tipo de componente especifica cómo el software AWS IoT Greengrass Core ejecuta el componente. Los componentes pueden tener los siguientes tipos:

- Núcleo (`aws.greengrass.nucleus`)

  El núcleo de Greengrass es el componente que proporciona la funcionalidad mínima del software AWS IoT Greengrass Core. Para obtener más información, consulte Núcleo de Greengrass.

- Complemento (`aws.greengrass.plugin`)

  El núcleo de Greengrass ejecuta un componente de complemento en la misma máquina virtual Java (JVM) que el núcleo. El núcleo se reinicia al cambiar la versión de un componente del complemento en un dispositivo principal. Para instalar y ejecutar los componentes del complemento, debe configurar el núcleo de Greengrass para que se ejecute como un servicio del sistema. Para obtener más información, consulte Configuración del núcleo de Greengrass como un servicio del sistema.

  Varios componentes proporcionados por AWS son componentes de complementos, lo que les permite interactuar directamente con el núcleo de Greengrass. Los componentes del complemento usan el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

- Genérico (`aws.greengrass.generic`)

  El núcleo de Greengrass ejecuta los scripts de ciclo de vida de un componente genérico si el componente define un ciclo de vida.

  Este tipo es el tipo predeterminado para los componentes personalizados.

- Lambda (`aws.greengrass.lambda`)

  El núcleo de Greengrass ejecuta un componente de función de Lambda mediante el componente lanzador de Lambda.

Cuando se crea un componente a partir de una función de Lambda, el componente tiene este tipo. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

> **Note**
>
> No le recomendamos que especifique el tipo de componente en una receta. AWS IoT Greengrass establece el tipo por usted al crear un componente.

Creación de componentes de AWS IoT Greengrass

Puede desarrollar componentes de AWS IoT Greengrass personalizados en una computadora de desarrollo local o en un dispositivo principal de Greengrass. AWS IoT Greengrass proporciona la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass (CLI de GDK) para ayudarlo a crear, compilar y publicar componentes a partir de plantillas de componentes predefinidas y componentes de comunidad. También puede ejecutar comandos de intérprete de comandos integrados para crear, compilar y publicar componentes. Elija entre las siguientes opciones para crear componentes de Greengrass personalizados:

- Uso de la CLI del kit de desarrollo de Greengrass

  Use la CLI de GDK para desarrollar componentes en una computadora de desarrollo local. La CLI de GDK crea y empaqueta el código de origen de los componentes en una receta y artefactos que puede publicar como un componente privado en el servicio de AWS IoT Greengrass. Puede configurar la CLI de GDK para que actualice automáticamente la versión del componente y los URI del artefacto al publicar el componente, de modo que no necesite actualizar la receta cada vez. Para desarrollar un componente mediante la CLI de GDK, puede partir de una plantilla o un componente de comunidad del catálogo de software de Greengrass. Para obtener más información, consulte Interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass.
- Ejecute los comandos de intérprete de comandos integrados

  Puede ejecutar comandos de intérprete de comandos integrados para desarrollar componentes en una computadora de desarrollo local o en un dispositivo principal de Greengrass. Los comandos de intérprete de comandos se usan para copiar o compilar el código de origen de los componentes en artefactos. Cada vez que cree una nueva versión de un componente, debe crear o actualizar la receta con la nueva versión del componente. Al publicar el componente en el servicio de AWS IoT Greengrass, debe actualizar el URI del artefacto de cada componente de la receta.

Temas

- Creación de un componente (CLI de GDK)
- Creación de un componente (comandos de intérprete de comandos)

Creación de un componente (CLI de GDK)

Siga las instrucciones de esta sección para crear y compilar un componente mediante la CLI de GDK.

Desarrollo de un componente de Greengrass (CLI de GDK)

1. Si aún no lo ha hecho, instale la CLI de GDK en su computadora de desarrollo. Para obtener más información, consulte Instalación o actualización de la interfaz de línea de comandos del kit de desarrollo de AWS IoT Greengrass.
2. Vaya a la carpeta en la que desea crear las carpetas de componentes.

   Linux or Unix

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```
3. Elija una plantilla de componentes o un componente de comunidad para descargar. La CLI de GDK descarga la plantilla o el componente de comunidad, por lo que puede empezar con un ejemplo funcional. Use el comando component list para recuperar la lista de plantillas o componentes de comunidad disponibles.
   - Para mostrar una lista de las plantillas de componentes, ejecute el siguiente comando. Cada línea de la respuesta incluye el nombre de la plantilla y el lenguaje de programación.

```
gdk component list --template
```

- Para mostrar una lista de componentes de comunidad, ejecute el siguiente comando.

```
gdk component list --repository
```

4. Cree y cambie a una carpeta de componentes en la que la CLI de GDK descargue la plantilla o el componente de comunidad. Reemplace *HelloWorld* por el nombre del componente u otro nombre que lo ayude a identificar esta carpeta de componentes.

   Linux or Unix

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   PowerShell

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

5. Descargue la plantilla o el componente de comunidad en la carpeta actual. Use el comando component init.

   - Para crear una carpeta de componentes a partir de una plantilla, ejecute el siguiente comando. Reemplace *HelloWorld* por el nombre de la plantilla y reemplace *python* por el nombre del lenguaje de programación.

     ```
     gdk component init --template HelloWorld --language python
     ```

   - Para crear una carpeta de componentes a partir de un componente de comunidad, ejecute el siguiente comando. Reemplace *ComponentName* por el nombre del componente de comunidad.

     ```
     gdk component init --repository ComponentName
     ```

> **Note**
>
> Si usa la versión 1.0.0 de la CLI de GDK, debe ejecutar este comando en una carpeta vacía. La CLI de GDK descarga la plantilla o el componente de comunidad en la carpeta actual.
> Si usa la versión 1.1.0 de la CLI de GDK o una versión posterior, puede especificar el argumento `--name` para especificar la carpeta en la que la CLI de GDK descarga la plantilla o el componente de comunidad. Si usa este argumento, especifique una carpeta que no existe. La CLI de GDK crea la carpeta por usted. Si no se especifica este argumento, la CLI de GDK usa la carpeta actual, que debe estar vacía.

6. La CLI de GDK lee el archivo de configuración de la CLI de GDK, denominado `gdk-config.json`, para crear y publicar componentes. Este archivo de configuración existe en la raíz de la carpeta del componente. El paso anterior crea este archivo por usted. En este paso, se actualiza `gdk-config.json` con información acerca de su componente. Haga lo siguiente:

   a. Abra `gdk-config.json` en un editor de texto.

   b. (Opcional) Cambie el nombre del componente. El nombre del componente es la clave del objeto `component`.

   c. Cambie el autor del componente.

   d. (Opcional) Cambie la versión del componente. Especifique uno de los siguientes valores:

      - `NEXT_PATCH`: al elegir esta opción, la CLI de GDK establece la versión al publicar el componente. La CLI de GDK consulta el servicio de AWS IoT Greengrass para identificar la versión más reciente publicada del componente. A continuación, establece la versión en la siguiente versión del parche posterior a esa versión. Si no ha publicado el componente antes, la CLI de GDK usa la versión `1.0.0`.

        Si elige esta opción, no podrá usar la CLI de Greengrass para implementar y probar localmente el componente en su computadora de desarrollo local que ejecuta el software AWS IoT Greengrass Core. Para habilitar las implementaciones locales, debe especificar una versión semántica en su lugar.

      - Una versión semántica, como **`1.0.0`**. Las versiones semánticas siguen un sistema de números de major.minor.patch. Para obtener más información, consulte la especificación semántica de la versión.

Si desarrolla componentes en un dispositivo principal de Greengrass en el que desee implementar y probar el componente, elija esta opción. Debe compilar el componente con una versión específica para crear implementaciones locales con la CLI de Greengrass.

e. (Opcional) Cambie la configuración de compilación del componente. La configuración de compilación define cómo la CLI de GDK compila el origen del componente en artefactos. Para `build_system`, puede elegir entre las siguientes opciones:

- `zip`: empaqueta la carpeta del componente en un archivo ZIP para definirla como el único artefacto del componente. Elija esta opción para los siguientes tipos de componentes:
  - Componentes que usan lenguajes de programación interpretados, como Python o JavaScript.
  - Componentes que empaquetan archivos distintos del código, como modelos de machine learning u otros recursos.

  La CLI de GDK comprime la carpeta del componente en un archivo zip con el mismo nombre que la carpeta del componente. Por ejemplo, si el nombre de la carpeta del componente es `HelloWorld`, la CLI de GDK crea un archivo zip denominado `HelloWorld.zip`.

  > **Note**
  >
  > Si usa la versión 1.0.0 de la CLI de GDK en un dispositivo Windows, los nombres de las carpetas y los archivos zip de los componentes deben contener solo letras minúsculas.

  Cuando la CLI de GDK comprime la carpeta del componente en un archivo zip, omite los siguientes archivos:

  - El archivo `gdk-config.json`
  - El archivo de receta (`recipe.json` o `recipe.yaml`)
  - Carpetas de compilación, como `greengrass-build`
- `maven`: ejecuta el comando `mvn clean package` para compilar el origen del componente en artefactos. Elija esta opción para los componentes que usan Maven, como los componentes de Java.

En dispositivos Windows, esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

- `gradle`: ejecuta el comando `gradle build` para compilar el origen del componente en artefactos. Elija esta opción para los componentes que usan Gradle. Esta característica está disponible para la versión 1.1.0 y posteriores de la CLI de GDK.

  El sistema de compilación `gradle` admite Kotlin DSL como archivo de compilación. Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

- `gradlew`: ejecuta el comando `gradlew` para compilar el origen del componente en artefactos. Elija esta opción para los componentes que usan el Gradle Wrapper.

  Esta característica está disponible para la versión 1.2.0 y versiones posteriores de la CLI de GDK.

- `custom`: ejecuta un comando personalizado para compilar el origen del componente en una receta y artefactos. Especifique el comando personalizado en el parámetro `custom_build_command`.

f. Si especifica `custom` para `build_system`, agregue `custom_build_command` al objeto `build`. En `custom_build_command`, especifique una sola cadena o lista de cadenas, donde cada cadena es una palabra del comando. Por ejemplo, para ejecutar un comando de compilación personalizado para un componente de C++, puede especificar **`["cmake", "--build", "build", "--config", "Release"]`**.

g. Si usa la versión 1.1.0 o posteriores de la CLI de GDK, puede especificar el argumento `--bucket` para especificar el bucket de S3 en el que la CLI de GDK carga los artefactos del componente. Si no especifica este argumento, la CLI de GDK se carga en el bucket de S3 cuyo nombre es *`bucket`*`-`*`region`*`-`*`accountId`*, donde *`bucket`* y *`region`* son los valores que especifica en `gdk-config.json` y *`acountId`* es el ID de su Cuenta de AWS. La CLI de GDK crea el bucket si no existe.

   Cambie la configuración de publicación del componente. Haga lo siguiente:

   i. Especifique el nombre del bucket de S3 que se usará para alojar artefactos de componentes.

   ii. Especifique la Región de AWS donde la CLI de GDK publica el componente.

Cuando haya terminado con este paso, el archivo `gdk-config.json` tendrá un aspecto similar al siguiente ejemplo.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. Actualice el archivo de receta del componente, denominado `recipe.yaml` o `recipe.json`. Haga lo siguiente:

   a. Si ha descargado una plantilla o un componente de comunidad que usa el sistema de compilación `zip`, compruebe que el nombre del artefacto zip coincide con el nombre de la carpeta del componente. La CLI de GDK comprime la carpeta del componente en un archivo zip con el mismo nombre que la carpeta del componente. La receta contiene el nombre del artefacto zip en la lista de artefactos de componentes y en los scripts de ciclo de vida que usan archivos del artefacto zip. Actualice las definiciones `Artifacts` y `Lifecycle` de forma que el nombre del archivo zip coincida con el nombre de la carpeta del componente. Los siguientes ejemplos de recetas parciales resaltan el nombre del archivo zip en las definiciones `Artifacts` y `Lifecycle`.

      JSON

      ```
      {
        ...
        "Manifests": [
          {
            "Platform": {
      ```

```
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      Run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

b. (Opcional) Actualice la descripción del componente, la configuración predeterminada, los artefactos, los scripts del ciclo de vida y el soporte de la plataforma. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.

Cuando haya terminado con este paso, el archivo de receta puede ser similar a los siguientes ejemplos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
```

```
  "ComponentName": "{COMPONENT_NAME}",
  "ComponentVersion": "{COMPONENT_VERSION}",
  "ComponentDescription": "This is a simple Hello World component written in
 Python.",
  "ComponentPublisher": "{COMPONENT_AUTHOR}",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "World"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
ComponentDescription: "This is a simple Hello World component written in
 Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:
```

```
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      Run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

8. Desarrolle y compile el componente de Greengrass. El comando component build produce una receta y artefactos en la carpeta `greengrass-build` de la carpeta del componente. Ejecute el siguiente comando.

```
gdk component build
```

Cuando esté listo para probar el componente, use la CLI de GDK para publicarlo en el servicio de AWS IoT Greengrass. A continuación, puede implementar el componente en los dispositivos principales de Greengrass. Para obtener más información, consulte Publique componentes para desplegarlos en sus dispositivos principales.

Creación de un componente (comandos de intérprete de comandos)

Siga las instrucciones de esta sección para crear carpetas de recetas y artefactos que contengan el código de origen y los artefactos de varios componentes.

Desarrollo de un componente de Greengrass (comandos de intérprete de comandos)

1. Cree una carpeta para sus componentes con subcarpetas para recetas y artefactos. Ejecute los siguientes comandos en su dispositivo principal de Greengrass para crear estas carpetas y cambiarlas a la carpeta de componentes. Reemplace *~/greengrassv2* o *%USERPROFILE%\greengrassv2* por la ruta a la carpeta que se usará para el desarrollo local.

   Linux or Unix

   ```
   mkdir -p ~/greengrassv2/{recipes,artifacts}
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
   ```

```
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. Use un editor de texto para crear un archivo de recetas que define los metadatos, parámetros, dependencias, ciclo de vida y capacidad de plataforma de su componente. Incluya la versión del componente en el nombre del archivo de recetas para poder identificar qué receta refleja qué versión del componente. Puede elegir el formato YAML o JSON para su receta.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   JSON

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   nano recipes/com.example.HelloWorld-1.0.0.yaml
   ```

   > **Note**
   >
   > AWS IoT Greengrass usa versiones semánticas para los componentes. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. Por ejemplo, la versión `1.0.0` representa el primer lanzamiento principal de un componente. Para obtener más información, consulte la especificación semántica de la versión.

3. Defina la receta de su componente. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.

   Es posible que la receta sea similar a la siguiente receta de ejemplo de Hello World.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
   ```

```
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
```

```
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Esta receta ejecuta un script de Python de Hello World, que puede ser similar al siguiente script de ejemplo.

```
import sys

message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

4. Cree una carpeta para desarrollar la versión del componente. Le recomendamos que use una carpeta independiente para los artefactos de cada versión del componente, de modo que pueda identificar los artefactos de cada versión del componente. Ejecute el siguiente comando.

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

   PowerShell

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

> **Important**
>
> Debe usar el siguiente formato para la ruta de la carpeta de artefactos. Incluya el nombre y la versión del componente que especifique en la receta.
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. Cree los artefactos de su componente en la carpeta creada en el paso anterior. Los artefactos pueden incluir software, imágenes y cualquier otro binario que use el componente.

   Cuando el componente esté listo, pruébelo.

Prueba de los componentes de AWS IoT Greengrass con implementaciones locales

Si desarrolla un componente de Greengrass en un dispositivo principal, puede crear una implementación local para instalarlo y probarlo. Siga los pasos de esta sección para crear una implementación local.

Si desarrolla el componente en un equipo diferente, como un equipo de desarrollo local, no podrá crear una implementación local. En su lugar, publique el componente en el servicio de AWS IoT Greengrass para poder implementarlo en los dispositivos principales de Greengrass para probarlo. Para obtener más información, consulte Publique componentes para desplegarlos en sus dispositivos principales y Implemente AWS IoT Greengrass componentes en los dispositivos.

Cómo probar un componente en un dispositivo principal de Greengrass

1. El dispositivo principal registra eventos como las actualizaciones de componentes. Puede ver este archivo de registro para detectar y solucionar errores en su componente, como una receta no válida. Este archivo de registro también muestra los mensajes que el componente imprime en formato estándar (stdout). Le recomendamos que abra una sesión de terminal adicional en su dispositivo principal para observar los nuevos mensajes de registro en tiempo real. Abra una nueva sesión de terminal, por ejemplo, mediante SSH, y ejecute el siguiente comando para ver los registros. Sustituya */greengrass/v2* por la ruta a la carpeta raíz de AWS IoT Greengrass.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

También puede ver el archivo de registro de su componente.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

2. En la sesión de terminal original, ejecute el siguiente comando para actualizar el dispositivo principal con su componente. Sustituya */greengrass/v2* por la ruta a la carpeta raíz de AWS IoT Greengrass y sustituya *~/greengrassv2* por la ruta a su carpeta de desarrollo local.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

> **Note**
>
> También puede usar el comando `greengrass-cli deployment create` para establecer el valor de los parámetros de configuración de su componente. Para obtener más información, consulte crear.

3. Utilice el comando `greengrass-cli deployment status` para supervisar el estado de la implementación del componente.

   Unix or Linux

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment status \
     -i deployment-id
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment status ^
     -i deployment-id
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment status `
     -i deployment-id
   ```

4. Pruebe el componente cuando se ejecuta en el dispositivo principal de Greengrass. Cuando termine esta versión del componente, puede cargarlo en el servicio de AWS IoT Greengrass. A continuación, puede implementar el componente en otros dispositivos principales. Para obtener más información, consulte Publique componentes para desplegarlos en sus dispositivos principales.

Publique componentes para desplegarlos en sus dispositivos principales

Después de crear o completar una versión de un componente, puede publicarla en el servicio de AWS IoT Greengrass. Luego, puede implementarlo en los dispositivos principales de Greengrass.

Si utiliza la CLI del kit de desarrollo de Greengrass (CLI del GDK) para desarrollar y crear un componente, puede utilizar la CLI de GDK para publicar el componente en la Nube de AWS. De lo contrario, utilice el intérprete de comandos integrados y la AWS CLI para publicar el componente.

También se puede utilizar AWS CloudFormation para crear componentes y otros recursos de AWS a partir de plantillas. Para obtener más información, consulte ¿Qué es AWS CloudFormation? y AWS::GreengrassV2::ComponentVersion en la Guía del usuario de AWS CloudFormation.

Temas

- Publicación de un componente (CLI de GDK)
- Publicación de un componente (intérprete de comandos)

Publicación de un componente (CLI de GDK)

Siga las instrucciones de esta sección para publicar un componente mediante la CLI de GDK. La CLI de GDK carga los artefactos de compilación en un bucket de S3, actualiza los URI de los artefactos en la receta y crea el componente a partir de la receta. Debe especificar el bucket y la región de S3 que se van a utilizar en el archivo de configuración de la CLI de GDK.

Si usa la versión 1.1.0 o posteriores de la CLI de GDK, puede especificar el argumento `--bucket` para especificar el bucket de S3 en el que la CLI de GDK carga los artefactos del componente. Si no especifica este argumento, la CLI de GDK se carga en el bucket de S3 cuyo nombre es *`bucket`*`-`*`region`*`-`*`accountId`*, donde *`bucket`* y *`region`* son los valores que especifica en `gdk-config.json` y *`acountId`* es el ID de su Cuenta de AWS. La CLI de GDK crea el bucket si no existe.

> **Important**
>
> Los roles de los dispositivos principales no permiten el acceso a los buckets de S3 de forma predeterminada. Si es la primera vez que utiliza este bucket de S3, debe agregar permisos al rol para que los dispositivos principales puedan recuperar los artefactos de los componentes de este bucket de S3. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.

Publicación de un componente de Greengrass (CLI de GDK)

1. Abra la carpeta del componente en una ventana del sistema o un terminal.
2. Si aún no lo ha hecho, cree el componente de Greengrass. El comando component build produce una receta y artefactos en la carpeta `greengrass-build` de la carpeta del componente. Ejecute el siguiente comando.

   ```
   gdk component build
   ```

3. Publique el componente en la Nube de AWS. El comando component publish carga los artefactos del componente en Amazon S3 y actualiza la receta del componente con el URI de cada artefacto. A continuación, crea el componente en el servicio de AWS IoT Greengrass.

   > **Note**
   >
   > AWS IoT Greengrass calcula el resumen de cada artefacto al crear el componente. Esto significa que no puede modificar los archivos de artefactos de su bucket de S3 después de crear un componente. Si lo hace, las implementaciones que incluyan este componente fallarán porque el resumen del archivo no coincide. Si modifica un archivo de artefactos, debe crear una nueva versión del componente.

   Si especifica `NEXT_PATCH` para la versión del componente en el archivo de configuración de la CLI del GDK, la CLI del GDK utilizará la siguiente versión del parche que aún no exista en el servicio de AWS IoT Greengrass.

   Ejecute el siguiente comando.

   ```
   gdk component publish
   ```

   El resultado indica la versión del componente que creó la CLI de GDK.

   Una vez publicado el componente, puede implementarlo en los dispositivos principales. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Publicación de un componente (intérprete de comandos)

Utilice el siguiente procedimiento para publicar un componente mediante los intérprete de comandos y la AWS Command Line Interface (AWS CLI). Al publicar un componente, puede hacer lo siguiente:

1. Publique artefactos de componentes en un bucket de S3.
2. Agregue el URI de Amazon S3 de cada artefacto a la receta del componente.
3. Cree una versión del componente en AWS IoT Greengrass a partir de la receta del componente.

> **Note**
>
> Cada versión de componente que cargue debe ser única. Asegúrese de cargar la versión del componente correcta, ya que no podrá editarla después de cargarla.

Puede seguir estos pasos para publicar un componente desde su computadora de desarrollo o su dispositivo principal de Greengrass.

Publicación de un componente (intérprete de comandos)

1. Si el componente usa una versión que existe en el servicio de AWS IoT Greengrass, debe cambiarla. Abra la receta en un editor de texto, incremente la versión y guarde el archivo. Elija una nueva versión que refleje los cambios que ha realizado en el componente.

   > **Note**
   >
   > AWS IoT Greengrass usa versiones semánticas para los componentes. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. Por ejemplo, la versión `1.0.0` representa el primer lanzamiento principal de un componente. Para obtener más información, consulte la especificación semántica de la versión.

2. Si el componente tiene artefactos, haga lo siguiente:

   a. Publique los artefactos del componente en un bucket de S3 en su Cuenta de AWS.

> **Tip**
>
> Le recomendamos que incluya el nombre y la versión del componente en la ruta al artefacto del bucket de S3. Este esquema de nomenclatura puede ayudarlo a conservar los artefactos que utilizan las versiones anteriores del componente, de forma que pueda seguir admitiendo las versiones anteriores del componente.

Ejecute el siguiente comando para publicar un archivo de artefacto en un bucket de S3. Sustituya amzn-s3-demo-bucket por el nombre del bucket y sustituya *artifacts/com.example.helloworld/1.0.0/artifact.py* por la ruta al archivo del artefacto.

```
aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

> **Important**
>
> Los roles de los dispositivos principales no permiten el acceso a los buckets de S3 de forma predeterminada. Si es la primera vez que utiliza este bucket de S3, debe agregar permisos al rol para que los dispositivos principales puedan recuperar los artefactos de los componentes de este bucket de S3. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.

b. Agregue una lista llamada `Artifacts` a la receta del componente si no está presente. La lista `Artifacts` aparece en cada manifiesto, donde se definen los requisitos del componente en cada plataforma compatible (o los requisitos predeterminados del componente para todas las plataformas).

c. Agregue cada artefacto a la lista de artefactos o actualice el URI de los artefactos existentes. El URI de Amazon S3 está compuesto por el nombre del bucket y la ruta al objeto artefacto del bucket. Los URI de Amazon S3 de sus artefactos deberían ser similares al siguiente ejemplo.

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

Después de completar estos pasos, la receta debería tener una lista `Artifacts` similar a la siguiente.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
MyGreengrassComponent/1.0.0/artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
  ]
}
```

> **Note**
>
> Puede agregar la opción `"Unarchive": "ZIP"` para que un artefacto ZIP configure el software AWS IoT Greengrass Core de forma que descomprima el artefacto cuando se implemente el componente.

YAML

```
...
Manifests:
  - Lifecycle:
      ...
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/MyGreengrassComponent/1.0.0/
artifact.py
```

```
        Unarchive: NONE
```

> **Note**
>
> Puede utilizar la opción `Unarchive: ZIP` para configurar el software AWS IoT Greengrass Core para descomprimir un artefacto ZIP cuando se implemente el componente. Para obtener más información sobre cómo utilizar los artefactos ZIP en un componente, consulte la variable de receta artifacts:decompressedPath.

Para obtener más información acerca de las recetas, consulte AWS IoT Greengrass referencia de recetas de componentes.

3. Utilice la consola de AWS IoT Greengrass para crear un componente a partir del archivo de recetas.

   Utilice el siguiente comando para crear un componente a partir del archivo de recetas. Este comando crea el componente y lo publica como un componente de AWS IoT Greengrass privado en su Cuenta de AWS. Sustituya *path/to/recipeFile* por la ruta del archivo de receta.

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://path/to/recipeFile
   ```

   Copie el `arn` de la respuesta para comprobar el estado del componente en el paso siguiente.

   > **Note**
   >
   > AWS IoT Greengrass calcula el resumen de cada artefacto al crear el componente. Esto significa que no puede modificar los archivos de artefactos de su bucket de S3 después de crear un componente. Si lo hace, las implementaciones que incluyan este componente fallarán porque el resumen del archivo no coincide. Si modifica un archivo de artefactos, debe crear una nueva versión del componente.

4. Cada componente del servicio de AWS IoT Greengrass tiene un estado. Ejecute el siguiente comando para confirmar el estado de la versión del componente que publique en este procedimiento. Sustituya *com.example.HelloWorld* y *1.0.0* por la versión del componente que desee consultar. Sustituya `arn` por el ARN del paso anterior.

```
aws greengrassv2 describe-component --arn "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorld:versions:1.0.0"
```

La operación devuelve una respuesta que contiene los metadatos del componente. Los metadatos contienen un objeto `status` que contiene el estado del componente y cualquier error, si corresponde.

Cuando el estado del componente es `DEPLOYABLE`, puede implementar el componente en los dispositivos. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Interacción con servicios de AWS

Los dispositivos principales de Greengrass usan certificados X.509 para conectarse a AWS IoT Core mediante protocolos de autenticación mutua de TLS. Estos certificados permiten a los dispositivos interactuar con AWS IoT sin credenciales de AWS, que normalmente constan de un ID de clave de acceso y de una clave de acceso secreta. Otros servicios de AWS requieren credenciales de AWS en lugar de certificados X.509 para llamar a operaciones de la API en los puntos de conexión del servicio. AWS IoT Core tiene un proveedor de credenciales que permite a los dispositivos utilizar su certificado X.509 para autenticar las solicitudes de AWS. El proveedor de credenciales de AWS IoT autentica los dispositivos mediante un certificado X.509 y emite credenciales de AWS en forma de un token de seguridad temporal con privilegios limitados. Los dispositivos pueden utilizar este token para firmar y autenticar cualquier solicitud de AWS. Esto elimina la necesidad de almacenar credenciales de AWS en los dispositivos principales de Greengrass. Para obtener más información, consulte Autorización de llamadas a los servicios de AWS en la Guía para desarrolladores de AWS IoT Core.

Para obtener las credenciales de AWS IoT, los dispositivos principales de Greengrass utilizan un alias de rol AWS IoT que apunta a un rol de IAM. Este rol de IAM se denomina rol de intercambio de token. El alias del rol y el rol de intercambio de token se crean al instalar el software AWS IoT Greengrass Core. Para especificar el alias de rol que utiliza un dispositivo principal, configure el parámetro `iotRoleAlias` del Núcleo de Greengrass.

El proveedor de credenciales de AWS IoT asume el rol de intercambio de token en su nombre para proporcionar credenciales de AWS a los dispositivos principales. Puede adjuntar las políticas de IAM adecuadas a este rol para permitir que sus dispositivos principales accedan a sus recursos de AWS, como los artefactos de componentes en buckets de S3. Para obtener más información acerca de

cómo configurar el rol de intercambio de token, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Los dispositivos principales de Greengrass almacenan credenciales de AWS en la memoria y, de forma predeterminada, las credenciales caducan después de una hora. Si el software AWS IoT Greengrass Core se reinicia, debe volver a obtener las credenciales. Puede utilizar la operación UpdateRoleAlias para configurar el periodo de validez de las credenciales.

AWS IoT Greengrass proporciona un componente público, el componente del servicio de intercambio de token, que puede definir como una dependencia en su componente personalizado para interactuar con los servicios de AWS. El servicio de intercambio de token proporciona al componente una variable de entorno, `AWS_CONTAINER_CREDENTIALS_FULL_URI`, que define el URI de un servidor local que proporciona las credenciales de AWS. Al crear un cliente del SDK de AWS, el cliente comprueba esta variable de entorno y se conecta al servidor local para recuperar las credenciales de AWS y las utiliza para firmar las solicitudes de la API. Esto le permite usar los SDK de AWS y otras herramientas para llamar a los servicios de AWS en sus componentes. Para obtener más información, consulte Servicio de intercambio de token.

> **Important**
>
> La compatibilidad para adquirir credenciales de AWS de esta manera se agregó a los SDK de AWS el 13 de julio de 2016. Su componente debe utilizar una versión del SDK de AWS creada en esa fecha o después. Para obtener más información, consulte Uso de un SDK de AWS compatible en la Guía para desarrolladores de Amazon Elastic Container Service.

Para adquirir credenciales de AWS en su componente personalizado, defina `aws.greengrass.TokenExchangeService` como una dependencia en la receta del componente. El siguiente ejemplo de receta define un componente que instala boto3 y ejecuta un script de Python que usa las credenciales de AWS del servicio de intercambio de token para enumerar los buckets de Amazon S3.

> **Note**
>
> Para ejecutar este ejemplo de componente, el dispositivo debe tener el permiso `s3:ListAllMyBuckets`. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list
 S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "Run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "Run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
```

```
ComponentDescription: A component that uses the token exchange service to list S3
 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      Run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
      Run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py
```

Este ejemplo de componente ejecuta el siguiente script de Python, `list_s3_buckets.py`, que muestra una lista de los buckets de Amazon S3.

```
import boto3
import os

try:
    print("Creating boto3 S3 client...")
    s3 = boto3.client('s3')
    print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
```

```
    print("Successfully listed S3 buckets")
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

Ejecución de un contenedor de Docker

Puede configurar AWS IoT Greengrass los componentes para que ejecuten un contenedor Docker a partir de imágenes almacenadas en las siguientes ubicaciones:

- Repositorios de imágenes públicos y privados en Amazon Elastic Container Registry (Amazon ECR)
- Repositorio público de Docker Hub
- Registro público de confianza de Docker
- Bucket de S3

En su componente personalizado, incluya el URI de la imagen de Docker como un artefacto para recuperar la imagen y ejecutarla en el dispositivo principal. En el caso de las imágenes de Amazon ECR y Docker Hub, puede utilizar el componente administrador de aplicaciones de Docker para descargar las imágenes y administrar las credenciales de los repositorios privados de Amazon ECR.

Temas

- Requisitos
- Ejecute un contenedor de Docker desde una imagen pública en Amazon ECR o Docker Hub
- Ejecución de un contenedor de Docker desde una imagen privada en Amazon ECR
- Ejecute un contenedor de Docker desde una imagen en Amazon S3
- Uso de la comunicación entre procesos en los componentes del contenedor de Docker
- Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux)
- Uso del administrador de flujos en los componentes del contenedor de Docker (Linux)

Requisitos

Para ejecutar un contenedor de Docker en un componente, necesita lo siguiente:

- Un dispositivo principal de Greengrass. Si no dispone de una, consulte Tutorial: Introducción a AWS IoT Greengrass V2.

- Versión 1.9.1 o posterior de Docker Engine instalada en el dispositivo principal de Greengrass. La versión 20.10 es la última versión que se ha comprobado que funciona con el software AWS IoT Greengrass Core. Debe instalar Docker directamente en el dispositivo principal antes de implementar componentes que ejecuten contenedores de Docker.

  > **Tip**
  >
  > También puede configurar el dispositivo principal para instalar Docker Engine cuando se instale el componente. Por ejemplo, el siguiente script de instalación instala Docker Engine antes de cargar la imagen de Docker. Este script de instalación funciona en distribuciones de Linux basadas en Debian, como Ubuntu. Si configura el componente para instalar Docker Engine con este comando, puede que tenga que configurar `RequiresPrivilege` en `true` en el script del ciclo de vida para ejecutar la instalación y los comandos `docker`. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.
  >
  > ```
  > apt-get install docker-ce docker-ce-cli containerd.io && docker load -i
  >  {artifacts:path}/hello-world.tar
  > ```

- El usuario del sistema que ejecute un componente contenedor de Docker debe tener permisos de raíz o administrador, o bien debe configurar Docker para que se ejecute como un usuario no de raíz o no administrador.
  - En los dispositivos Linux, debe agregar un usuario al grupo de `docker` para llamar comandos `docker` sin `sudo`.
  - En los dispositivos Windows, puede agregar un usuario al grupo de `docker-users` para llamar comandos `docker` sin privilegios de administrador.

  Linux or Unix

  Para agregar `ggc_user`, o el usuario no raíz que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker`, ejecute el siguiente comando.

  ```
  sudo usermod -aG docker ggc_user
  ```

  Para obtener más información, consulte Administrar Docker como un usuario no raíz.

Windows Command Prompt (CMD)

Para agregar `ggc_user`, o el usuario que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker-users`, ejecute el siguiente comando como un administrador.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Para agregar `ggc_user`, o el usuario que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker-users`, ejecute el siguiente comando como un administrador.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Los archivos a los que accede el componente contenedor de Docker se montan como un volumen en el contenedor de Docker.
- Si configura el software AWS IoT Greengrass Core para usar un proxy de red, debe configurar Docker para que use el mismo servidor proxy.

Además de estos requisitos, también debe cumplir con los siguientes requisitos si se aplican a su entorno:

- Para usar Docker Compose para crear e iniciar sus contenedores de Docker, instale Docker Compose en su dispositivo principal de Greengrass y cargue el archivo de Docker Compose en un bucket de S3. Debe almacenar el archivo Compose en un depósito de S3 en el mismo componente Cuenta de AWS y Región de AWS como él. Para ver un ejemplo en el que se usa el comando `docker-compose up` en un componente personalizado, consulte Ejecute un contenedor de Docker desde una imagen pública en Amazon ECR o Docker Hub.
- Si utilizas un proxy AWS IoT Greengrass de red, configura el daemon de Docker para que utilice un servidor proxy.
- Si sus imágenes de Docker están almacenadas en Amazon ECR o Docker Hub, incluya el componente administrador de componentes de Docker como una dependencia en su componente de contenedor de Docker. Debe iniciar el daemon Docker en el dispositivo principal antes de implementar su componente.

Además, incluya la imagen URIs como elemento componente. La imagen URIs debe tener el formato `docker:`*`registry`*`/`*`image`*`[`*`:tag|@digest`*`]` que se muestra en los siguientes ejemplos:

- Imagen privada de Amazon ECR: `docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`
- Imagen pública de Amazon ECR: `docker:public.ecr.aws/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`
- Imagen pública de Docker Hub: `docker:`*`name`*`[`*`:tag|@digest`*`]`

Para obtener más información sobre cómo ejecutar contenedores de Docker a partir de imágenes almacenadas en repositorios públicos, consulte Ejecute un contenedor de Docker desde una imagen pública en Amazon ECR o Docker Hub.

- Si sus imágenes de Docker están almacenadas en un registro privado de Amazon ECR, debe incluir el componente del servicio de intercambio de token como una dependencia en el componente contenedor de Docker. Además, el rol de dispositivo de Greengrass debe permitir las acciones `ecr:GetAuthorizationToken`, `ecr:BatchGetImage` y `ecr:GetDownloadUrlForLayer` como se muestra en el siguiente ejemplo de política de IAM.

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Para obtener más información sobre cómo ejecutar contenedores de Docker a partir de imágenes almacenadas en repositorios privados de Amazon ECR, consulte Ejecución de un contenedor de Docker desde una imagen privada en Amazon ECR.

- Para usar imágenes de Docker almacenadas en un repositorio privado de Amazon ECR, el repositorio privado debe estar en el Región de AWS mismo lugar que el dispositivo principal.
- Si sus imágenes de Docker o archivos de Compose se almacenan en un bucket de S3, el rol del dispositivo de Greengrass debe conceder el permiso `s3:GetObject` para que los dispositivos principales descarguen las imágenes como artefactos de componentes, como se muestra en el siguiente ejemplo de política de IAM.

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Action": [
          "s3:GetObject"
        ],
        "Resource": [
          "*"
        ],
        "Effect": "Allow"
      }
    ]
  }
  ```

  Para obtener más información sobre cómo ejecutar contenedores de Docker a partir de imágenes almacenadas en Amazon S3, consulte Ejecute un contenedor de Docker desde una imagen en Amazon S3.

- Para usar la comunicación entre procesos (IPC), las credenciales de AWS o el administrador de flujos en el componente contenedor de Docker, debe especificar opciones adicionales al ejecutar el contenedor de Docker. Para obtener más información, consulte los siguientes temas:
  - Uso de la comunicación entre procesos en los componentes del contenedor de Docker
  - Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux)
  - Uso del administrador de flujos en los componentes del contenedor de Docker (Linux)

Ejecute un contenedor de Docker desde una imagen pública en Amazon ECR o Docker Hub

En esta sección, se describe cómo crear un componente personalizado que utilice Docker Compose para ejecutar un contenedor de Docker a partir de imágenes de Docker almacenadas en Amazon ECR y Docker Hub.

Cómo ejecutar un contenedor de Docker mediante Docker Compose

1. Cree y cargue un archivo de Docker Compose en un bucket de Amazon S3. Asegúrese de que el rol de dispositivo de Greengrass otorgue el permiso `s3:GetObject` para permitir que el dispositivo acceda al archivo de Compose. El archivo Compose de ejemplo que se muestra en el siguiente ejemplo incluye la imagen de Amazon CloudWatch Agent de Amazon ECR y la imagen de MySQL de Docker Hub.

   ```
   version: "3"
   services:
     cloudwatchagent:
       image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
     mysql:
       image: "mysql:8.0"
   ```

2. Cree un componente personalizado en su dispositivo AWS IoT Greengrass principal. La receta de ejemplo que se muestra en el ejemplo siguiente incluye las siguientes propiedades:

   - El componente del administrador de aplicaciones de Docker como dependencia. Este componente permite a AWS IoT Greengrass descargar imágenes de los repositorios públicos de Amazon ECR y Docker Hub.
   - Un artefacto componente que especifica una imagen de Docker en un repositorio público de Amazon ECR.
   - Un artefacto componente que especifica una imagen de Docker en un repositorio público de Docker Hub.
   - Un artefacto componente que especifica el archivo de Docker Compose que incluye los contenedores para las imágenes de Docker que quiere ejecutar.
   - Un script de ejecución de ciclo de vida que usa docker-compose up para crear e iniciar un contenedor a partir de las imágenes especificadas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyDockerComposeComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses Docker Compose to run images
 from public Amazon ECR and Docker Hub.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "Run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        Run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://amzn-s3-demo-bucket/folder/docker-compose.yaml"
```

> **Note**
>
> Para usar la comunicación entre procesos (IPC), las credenciales de AWS o el administrador de flujos en el componente contenedor de Docker, debe especificar opciones adicionales al ejecutar el contenedor de Docker. Para obtener más información, consulte los siguientes temas:
>
> - Uso de la comunicación entre procesos en los componentes del contenedor de Docker
> - Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux)
> - Uso del administrador de flujos en los componentes del contenedor de Docker (Linux)

3. Pruebe el componente para comprobar que funciona según lo previsto.

> **Important**
>
> Debe instalar e iniciar el daemon Docker antes de implementar su componente.

Tras implementar el componente localmente, puede ejecutar el comando docker container ls para comprobar que el contenedor se ejecuta.

```
docker container ls
```

4. Cuando el componente esté listo, cárguelo AWS IoT Greengrass para implementarlo en otros dispositivos principales. Para obtener más información, consulte Publique componentes para desplegarlos en sus dispositivos principales.

Ejecución de un contenedor de Docker desde una imagen privada en Amazon ECR

En esta sección, se describe cómo crear un componente personalizado que ejecute un contenedor de Docker desde una imagen de Docker almacenada en un repositorio privado en Amazon ECR.

Cómo ejecutar un contenedor de Docker

1. Cree un componente personalizado en su dispositivo AWS IoT Greengrass principal. Utilice la siguiente receta de ejemplo, que incluye las siguientes propiedades:

   - El componente del administrador de aplicaciones de Docker como dependencia. Este componente permite a AWS IoT Greengrass administrar las credenciales para descargar imágenes de repositorios privados.
   - El componente del servicio de intercambio de token como dependencia. Este componente permite AWS IoT Greengrass recuperar AWS credenciales para interactuar con Amazon ECR.
   - Un artefacto componente que especifica una imagen de Docker en un repositorio público de Amazon ECR.
   - Un script de ejecución de ciclo de vida que utiliza docker run para crear e iniciar un contenedor desde la imagen.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyPrivateDockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from a
 private Amazon ECR image.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
```

```
    },
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "Run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
 Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
```

```
        Run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]"
```

> **Note**
>
> Para usar la comunicación entre procesos (IPC), las credenciales de AWS o el administrador de flujos en el componente contenedor de Docker, debe especificar opciones adicionales al ejecutar el contenedor de Docker. Para obtener más información, consulte los siguientes temas:
>
> - Uso de la comunicación entre procesos en los componentes del contenedor de Docker
> - Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux)
> - Uso del administrador de flujos en los componentes del contenedor de Docker (Linux)

2. Pruebe el componente para comprobar que funciona según lo previsto.

> **Important**
>
> Debe instalar e iniciar el daemon Docker antes de implementar su componente.

Tras implementar el componente localmente, puede ejecutar el comando docker container ls para comprobar que el contenedor se ejecuta.

```
docker container ls
```

3. Cargue el componente AWS IoT Greengrass para implementarlo en otros dispositivos principales. Para obtener más información, consulte Publique componentes para desplegarlos en sus dispositivos principales.

Ejecute un contenedor de Docker desde una imagen en Amazon S3

En esta sección, se describe cómo ejecutar un contenedor de Docker en un componente desde una imagen de Docker almacenada en Amazon S3.

Cómo ejecutar un contenedor de Docker en un componente desde una imagen en Amazon S3

1. Ejecute el comando docker save para crear una copia de seguridad de un contenedor de Docker. Esta copia de seguridad se proporciona como un artefacto componente en el que ejecutar el contenedor en AWS IoT Greengrass. *hello-world*Sustitúyalo por el nombre de la imagen y *hello-world.tar* sustitúyalo por el nombre del archivo comprimido que se va a crear.

   ```
   docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
   ```

2. Cree un componente personalizado en su dispositivo AWS IoT Greengrass principal. Utilice la siguiente receta de ejemplo, que incluye las siguientes propiedades:

   - Un script de instalación del ciclo de vida que usa docker load para cargar una imagen de Docker desde un archivo.
   - Un script de ejecución de ciclo de vida que utiliza docker run para crear e iniciar un contenedor desde la imagen. La opción `--rm` limpia el contenedor cuando sale.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": {
             "Script": "docker load -i {artifacts:path}/hello-world.tar"
           },
           "Run": {
             "Script": "docker run --rm hello-world"
           }
         }
   ```

```
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      Run:
        Script: docker run --rm hello-world
```

> **Note**
>
> Para usar la comunicación entre procesos (IPC), las credenciales de AWS o el administrador de flujos en el componente contenedor de Docker, debe especificar opciones adicionales al ejecutar el contenedor de Docker. Para obtener más información, consulte los siguientes temas:
>
> - Uso de la comunicación entre procesos en los componentes del contenedor de Docker
> - Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux)
> - Uso del administrador de flujos en los componentes del contenedor de Docker (Linux)

3. Pruebe el componente para comprobar que funciona según lo previsto.

   Tras implementar el componente localmente, puede ejecutar el comando docker container ls para comprobar que el contenedor se ejecuta.

```
docker container ls
```

4. Cuando el componente esté listo, sube el archivo de imágenes de Docker a un bucket de S3 y agregue su URI a la receta del componente. A continuación, puede cargar el componente AWS IoT Greengrass para implementarlo en otros dispositivos principales. Para obtener más información, consulte Publique componentes para desplegarlos en sus dispositivos principales.

   Al terminar, la receta del componente tendrá un aspecto semejante al de este ejemplo.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from an
 image in an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": {
          "Script": "docker load -i {artifacts:path}/hello-world.tar"
        },
        "Run": {
          "Script": "docker run --rm hello-world"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar"
        }
      ]
    }
  ]
}
```

   YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      Run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

Uso de la comunicación entre procesos en los componentes del contenedor de Docker

Puede utilizar la biblioteca de comunicación entre procesos (IPC) de Greengrass SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes de Greengrass y. AWS IoT Core Para obtener más información, consulte Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core.

Para usar el IPC en un componente de contenedor de Docker, debe ejecutar el contenedor de Docker con los siguientes parámetros:

- Monte el socket IPC en el contenedor. El núcleo de Greengrass proporciona la ruta del archivo del socket IPC en la variable de entorno `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`.
- Establezca las variables de entorno `SVCUID` y `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` en los valores que el núcleo de Greengrass proporciona a los componentes. Su componente utiliza estas variables de entorno para autenticar las conexiones al núcleo de Greengrass.

Example Receta de ejemplo: publicar un mensaje MQTT en AWS IoT Core (Python)

La siguiente receta define un ejemplo de componente contenedor de Docker en el que se publica un mensaje MQTT. AWS IoT Core Esta receta tiene las siguientes propiedades:

- Una política de autorización (`accessControl`) que permite al componente publicar mensajes MQTT AWS IoT Core sobre todos los temas. Para obtener más información, consulte Autorización de los componentes para realizar operaciones de IPC y Autorización de IPC en AWS IoT Core MQTT.
- Un artefacto de componente que especifica una imagen de Docker como un archivo TAR en Amazon S3.
- Un script de instalación del ciclo de vida que carga la imagen de Docker desde el archivo TAR.
- Un script de ejecución de ciclo de vida que ejecuta un contenedor de Docker desde la imagen. El comando Docker run tiene los siguientes argumentos:
  - El argumento `-v` monta el conector IPC de Greengrass en el contenedor.
  - Los dos primeros argumentos `-e` establecen las variables de entorno necesarias en el contenedor de Docker.
  - Los argumentos `-e` adicionales establecen las variables de entorno utilizadas en este ejemplo.
  - El argumento `--rm` limpia el contenedor al salir.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
 message to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
            "policyDescription": "Allows access to publish to IoT Core on all
 topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
```

```
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "Run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC=
\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
```

```
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
          resources:
            - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      Run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux)

Puede usar el componente del servicio de intercambio de fichas para interactuar con AWS los servicios de los componentes de Greengrass. Este componente proporciona las credenciales de AWS del rol de intercambio de token del dispositivo principal mediante un servidor contenedor local. Para obtener más información, consulte Interacción con servicios de AWS.

> **Note**
>
> El ejemplo de esta sección solo funciona en los dispositivos principales de Linux.

Para usar AWS las credenciales del servicio de intercambio de fichas en un componente de contenedor de Docker, debe ejecutar el contenedor de Docker con los siguientes parámetros:

- Proporcione acceso a la red host mediante el argumento `--network=host`. Esta opción permite que el contenedor de Docker se conecte al servicio de intercambio de tokens local para recuperar las credenciales. AWS Este argumento solo funciona en Docker para Linux.

  > **Warning**
  >
  > Esta opción proporciona al contenedor acceso a todas las interfaces de red locales del host, por lo que es menos segura que si se ejecutan contenedores de Docker sin este acceso a la red del host. Tenga esto en cuenta al desarrollar y ejecutar componentes de contenedores de Docker que utilizan esta opción. Para obtener más información, consulte Red: host en la documentación de Docker.

- Establezca las variables `AWS_CONTAINER_CREDENTIALS_FULL_URI` y de `AWS_CONTAINER_AUTHORIZATION_TOKEN` entorno en los valores que el núcleo de Greengrass proporciona a los componentes. AWS SDKs utilice estas variables de entorno para recuperar las AWS credenciales.

Example Receta de ejemplo: enumerar buckets de S3 en un componente contenedor de Docker (Python)

La siguiente receta define un ejemplo de componente de contenedor de Docker que muestra los buckets de S3 de su Cuenta de AWS. Esta receta tiene las siguientes propiedades:

- El componente del servicio de intercambio de token como dependencia. Esta dependencia permite al componente recuperar AWS credenciales para interactuar con otros AWS servicios.
- Un artefacto componente que especifica una imagen de Docker como un archivo tar en Amazon S3.
- Un script de instalación del ciclo de vida que carga la imagen de Docker desde el archivo TAR.
- Un script de ejecución de ciclo de vida que ejecuta un contenedor de Docker desde la imagen. El comando Docker run tiene los siguientes argumentos:
  - El argumento `--network=host` proporciona al contenedor acceso a la red de host, de modo que el contenedor puede conectarse al servicio de intercambio de token.
  - El argumento `-e` establece las variables de entorno necesarias en el contenedor de Docker.
  - El argumento `--rm` limpia el contenedor al salir.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3
 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
        "Run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
 AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
```

```
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      Run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
          --rm list-s3-buckets
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

Uso del administrador de flujos en los componentes del contenedor de Docker (Linux)

Puede usar el componente administrador de flujos para administrar flujos de datos en los componentes de Greengrass. Este componente le permite procesar flujos de datos y transferir datos de IoT de gran volumen al Nube de AWS. AWS IoT Greengrass proporciona un SDK de administrador de transmisiones que se utiliza para interactuar con el componente de administrador de transmisiones. Para obtener más información, consulte Administración de flujos de datos en los dispositivos principales de Greengrass.

> **Note**
>
> El ejemplo de esta sección solo funciona en los dispositivos principales de Linux.

Para utilizar el SDK del administrador de flujos en un componente de contenedor de Docker, debe ejecutar el contenedor de Docker con los siguientes parámetros:

- Proporcione acceso a la red host mediante el argumento `--network=host`. Esta opción permite que el contenedor de Docker interactúe con el componente del administrador de flujos a través de una conexión TLS local. Este argumento solo funciona en Docker para Linux

> **Warning**
>
> Esta opción proporciona al contenedor acceso a todas las interfaces de red locales del host, por lo que es menos segura que si se ejecutan contenedores de Docker sin este acceso a la red del host. Tenga esto en cuenta al desarrollar y ejecutar componentes de contenedores de Docker que utilizan esta opción. Para obtener más información, consulte Red: host en la documentación de Docker.

- Si configura el componente del administrador de flujos para que requiera autenticación, que es el comportamiento predeterminado, establezca la variable de entorno `AWS_CONTAINER_CREDENTIALS_FULL_URI` en el valor que el núcleo de Greengrass proporciona a los componentes. Para obtener más información, consulte la configuración del administrador de flujos.
- Si configura el componente del administrador de flujos para que utilice un puerto que no sea el predeterminado, utilice la comunicación entre procesos (IPC) para obtener el puerto desde la configuración del componente administrador de flujos. Debe ejecutar el contenedor de Docker con opciones adicionales para usar el IPC. Para obtener más información, consulte los siguientes temas:
  - Conexión al administrador de flujos en el código de la aplicación
  - Uso de la comunicación entre procesos en los componentes del contenedor de Docker

Example Receta de ejemplo: transmitir un archivo a un bucket de S3 en un componente contenedor de Docker (Python)

La siguiente receta define un ejemplo de componente contenedor de Docker que crea un archivo y lo transmite a un bucket de S3. Esta receta tiene las siguientes propiedades:

- El componente del administrador de flujos como dependencia. Esta dependencia permite que el componente utilice el SDK del administrador de flujos para interactuar con el componente administrador de flujos.
- Un artefacto de componente que especifica una imagen de Docker como un archivo TAR en Amazon S3.
- Un script de instalación del ciclo de vida que carga la imagen de Docker desde el archivo TAR.
- Un script de ejecución de ciclo de vida que ejecuta un contenedor de Docker desde la imagen. El comando Docker run tiene los siguientes argumentos:

- El argumento `--network=host` proporciona al contenedor acceso a la red host, de modo que el contenedor puede conectarse al componente administrador de flujos.
- El primer argumento `-e` establece la variable de entorno `AWS_CONTAINER_AUTHORIZATION_TOKEN` requerida en el contenedor de Docker.
- Los argumentos `-e` adicionales establecen las variables de entorno utilizadas en este ejemplo.
- El argumento `-v` monta la carpeta de trabajo del componente en el contenedor. En este ejemplo, se crea un archivo en la carpeta de trabajo para subirlo a Amazon S3 mediante el administrador de flujos.
- El argumento `--rm` limpia el contenedor al salir.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.StreamFileToS3",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Creates a text file and uses stream manager to stream the
 file to S3.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "bucketName": ""
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
        "Run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
 -e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
 {work:path}:{work:path} --rm stream-file-to-s3"
```

```
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      Run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e BUCKET_NAME="{configuration:/bucketName}" \
          -e WORK_PATH="{work:path}" \
          -v {work:path}:{work:path} \
          --rm stream-file-to-s3
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass referencia de recetas de componentes

La receta del componente es un archivo que define los detalles, las dependencias, los artefactos y los ciclos de vida de un componente. El ciclo de vida del componente especifica los comandos que se deben ejecutar para instalar, ejecutar y apagar el componente, por ejemplo. El AWS IoT Greengrass núcleo utiliza los ciclos de vida que usted define en la receta para instalar y ejecutar los componentes. El AWS IoT Greengrass servicio utiliza la receta para identificar las dependencias y los artefactos que se van a implementar en los dispositivos principales al implementar el componente.

En la receta, puede definir dependencias y ciclos de vida únicos para cada plataforma compatible con un componente. Puede usar esta capacidad para implementar un componente en dispositivos con varias plataformas que tengan requisitos diferentes. También puedes usarlo para AWS IoT Greengrass evitar que se instale un componente en dispositivos que no lo admitan.

Cada receta contiene una lista de manifiestos. Cada manifiesto especifica un conjunto de requisitos de plataforma, así como el ciclo de vida y los artefactos que usarán los dispositivos principales cuya plataforma cumpla con esos requisitos. El dispositivo principal usa el primer manifiesto con los requisitos de plataforma que cumple el dispositivo. Especifique un manifiesto sin ningún requisito de plataforma que coincida con cualquier dispositivo principal.

También puede especificar un ciclo de vida global que no esté en un manifiesto. En el ciclo de vida global, puede usar claves de selección que identifiquen las subsecciones del ciclo de vida. Luego, puede especificar estas claves de selección en un manifiesto para usar esas secciones del ciclo de vida global además del ciclo de vida del manifiesto. El dispositivo principal usa las claves de selección del manifiesto solo si el manifiesto no define un ciclo de vida. Puede usar la selección `all` de un manifiesto para hacer coincidir las secciones del ciclo de vida global sin claves de selección.

Una vez que el software AWS IoT Greengrass principal selecciona un manifiesto que coincide con el dispositivo principal, hace lo siguiente para identificar los pasos del ciclo de vida que se deben seguir:

- Si el manifiesto seleccionado define un ciclo de vida, el dispositivo principal usa ese ciclo de vida.
- Si el manifiesto seleccionado no define un ciclo de vida, el dispositivo principal usa el ciclo de vida global. El dispositivo principal hace lo siguiente para identificar qué secciones del ciclo de vida global debe usar:
  - Si el manifiesto define las claves de selección, el dispositivo principal usa las secciones del ciclo de vida global que contienen las claves de selección del manifiesto.

- Si el manifiesto no define las claves de selección, el dispositivo principal usa las secciones del ciclo de vida global que no tienen claves de selección. Este comportamiento equivale a un manifiesto que define la selección `all`.

> **Important**
>
> Un dispositivo principal debe cumplir como mínimo los requisitos de plataforma de un manifiesto para instalar el componente. Si ningún manifiesto coincide con el dispositivo principal, el software AWS IoT Greengrass principal no instala el componente y se produce un error en la implementación.

Puede definir recetas en formato JSON o YAML. La sección de ejemplos de recetas incluye recetas en cada formato.

Temas

- Validación de receta
- Formato de receta
- Variables de receta
- Ejemplos de receta

Validación de receta

Greengrass valida la receta de un componente JSON o YAML al crear una versión del componente. Esta validación de recetas comprueba la receta de sus componentes JSON o YAML para detectar errores comunes a fin de evitar posibles problemas de implementación. La validación comprueba la receta para detectar errores comunes (p. ej., falta de comas, corchetes y campos) y se asegura de que la receta esté bien formada.

Si recibe un mensaje de error al validar una receta, compruebe si en ella faltan comas, corchetes o campos. Compruebe que no le falte ningún campo consultando el formato de la receta.

Formato de receta

Al definir una receta para un componente, se especifica la siguiente información en el documento de receta. La misma estructura se aplica a las recetas en los formatos YAML y JSON.

`RecipeFormatVersion`

La versión de plantilla para la receta. Elija la opción siguiente:

- `2020-01-25`

`ComponentName`

El nombre del componente que define esta receta. El nombre del componente debe ser único Cuenta de AWS en cada región.

> **Consejos**
>
> - Use el formato de nombre de dominio inverso para evitar colisiones de nombres dentro de su empresa. Por ejemplo, si su empresa es propietaria de `example.com` y usted trabaja en un proyecto de energía solar, puede nombrar a su componente Hello World `com.example.solar.HelloWorld`. Esto ayuda a evitar colisiones entre los nombres de los componentes dentro de su empresa.
> - Evite el prefijo `aws.greengrass` en los nombres de sus componentes. AWS IoT Greengrass usa este prefijo para los componentes públicos que proporciona. Si elige el mismo nombre que un componente público, su componente reemplazará a ese componente. A continuación, AWS IoT Greengrass proporciona su componente en lugar del componente público cuando despliega componentes que dependen de ese componente público. Esta característica le permite anular el comportamiento de los componentes públicos, pero también puede interrumpir otros componentes si no tiene intención de anular un componente público.

`ComponentVersion`

Esta es la versión del componente. El valor máximo para los valores principales, secundarios y de parche es 999999.

> **Note**
>
> AWS IoT Greengrass usa versiones semánticas para los componentes. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. Por ejemplo, la versión `1.0.0` representa el primer lanzamiento principal de un componente. Para obtener más información, consulte la especificación semántica de la versión.

`ComponentDescription`

(Opcional) La descripción del componente.

`ComponentPublisher`

El publicador o el autor del componente.

`ComponentConfiguration`

(Opcional) Un objeto que define la configuración o los parámetros del componente. Defina la configuración predeterminada y, a continuación, al implementar el componente, puede especificar el objeto de configuración que se va a proporcionar al componente. La configuración de los componentes admite parámetros y estructuras anidados. Este objeto contiene la siguiente información:

`DefaultConfiguration`

Un objeto que define la configuración predeterminada del componente. Usted define la estructura de este objeto.

> **Note**
>
> AWS IoT Greengrass usa JSON para los valores de configuración. JSON especifica un tipo de número, pero no diferencia entre números enteros y flotantes. Como resultado, los valores de configuración pueden convertirse en valores flotantes en AWS IoT Greengrass. Para garantizar que su componente use el tipo de datos correcto, le recomendamos que defina los valores de configuración numéricos como cadenas. A continuación, pida a su componente que los analice como enteros o flotantes. Esto garantiza que los valores de configuración sean del mismo tipo en la configuración y en el dispositivo principal.

`ComponentDependencies`

(Opcional) Un diccionario de objetos, cada uno de los cuales define una dependencia de un componente para el componente. La clave de cada objeto identifica el nombre de la dependencia del componente. AWS IoT Greengrass instala las dependencias de los componentes cuando se instala el componente. AWS IoT Greengrass espera a que se inicien las dependencias antes de iniciar el componente. Cada objeto contiene la siguiente información:

`VersionRequirement`

La restricción de versión semántica de estilo npm que define las versiones de los componentes compatibles para esta dependencia. Puede especificar una versión o un rango de versiones. Para obtener más información, consulte calculador de versión semántica de npm.

`DependencyType`

(Opcional) El tipo de esta dependencia. Puede elegir entre las siguientes opciones.

- `SOFT` — El componente no se reinicia si la dependencia cambia de estado.
- `HARD` — El componente se reinicia si la dependencia cambia de estado.

El valor predeterminado es `HARD`.

`ComponentType`

(Opcional) El tipo de componente.

> **Note**
>
> No se recomienda especificar el tipo de componente en una receta. AWS IoT Greengrass establece el tipo automáticamente al crear un componente.

Puede tratarse de uno de los siguientes tipos:

- `aws.greengrass.generic`: el componente ejecuta comandos o proporciona artefactos.
- `aws.greengrass.lambda`: el componente ejecuta una función de Lambda mediante el componente lanzador de Lambda. El parámetro `ComponentSource` especifica el ARN de la función de Lambda que ejecuta este componente.

  No se recomienda utilizar esta opción, ya que se establece AWS IoT Greengrass al crear un componente a partir de una función Lambda. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.
- `aws.greengrass.plugin`: el componente se ejecuta en la misma máquina virtual de Java (JVM) que el núcleo de Greengrass. Si implementa o reinicia un componente del complemento, el núcleo de Greengrass se reinicia.

Los componentes del complemento usan el mismo archivo de registro que el núcleo de Greengrass. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

No se recomienda utilizar esta opción en las recetas de componentes, ya que está AWS pensada para componentes proporcionados y escritos en Java que interactúan directamente con el núcleo de Greengrass. Para obtener más información sobre qué componentes públicos son complementos, consulte Componentes proporcionados por AWS.

- `aws.greengrass.nucleus`: el componente del núcleo. Para obtener más información, consulte Núcleo de Greengrass.

  No le recomendamos que use esta opción en las recetas de componentes. Está diseñado para el componente núcleo de Greengrass, que proporciona la funcionalidad mínima del software AWS IoT Greengrass Core.

El valor predeterminado es `aws.greengrass.generic` cuando se crea un componente a partir de una receta o `aws.greengrass.lambda` cuando se crea un componente a partir de una función de Lambda.

Para obtener más información, consulte Tipos de componentes.

`ComponentSource`

(Opcional) El ARN de la función de Lambda que ejecuta un componente.

No se recomienda especificar la fuente del componente en una receta. AWS IoT Greengrass establece este parámetro cuando crea un componente a partir de una función Lambda. Para obtener más información, consulte Ejecución de funciones de AWS Lambda.

`Manifests`

Una lista de objetos, cada uno de los cuales define el ciclo de vida, los parámetros y los requisitos del componente para una plataforma. Si un dispositivo principal cumple con los requisitos de plataforma de varios manifiestos, AWS IoT Greengrass utiliza el primer manifiesto que coincida con el dispositivo principal. Para garantizar que los dispositivos principales usen el manifiesto correcto, defina primero los manifiestos con requisitos de plataforma más estrictos. Un manifiesto que se aplique a todas las plataformas debe ser el último manifiesto de la lista.

> **Important**
>
> Un dispositivo principal debe cumplir como mínimo los requisitos de plataforma de un manifiesto para instalar el componente. Si ningún manifiesto coincide con el dispositivo principal, el software AWS IoT Greengrass principal no instala el componente y se produce un error en la implementación.

Cada objeto contiene la siguiente información:

`Name`

(Opcional) Un nombre fácil de recordar para la plataforma que define este manifiesto.

Si omite este parámetro, AWS IoT Greengrass crea un nombre a partir de la plataforma `os` `yarchitecture`.

`Platform`

(Opcional) Un objeto que define la plataforma a la que se aplica este manifiesto. Omita este parámetro para definir un manifiesto que se aplique a todas las plataformas.

Este objeto especifica los pares clave-valor sobre la plataforma en la que se ejecuta un dispositivo principal. Al implementar este componente, el software AWS IoT Greengrass Core compara estos pares clave-valor con los atributos de la plataforma en el dispositivo principal. El software AWS IoT Greengrass principal siempre define `os` `yarchitecture`, y puede definir, atributos adicionales. Puede especificar atributos de plataforma personalizados para un dispositivo principal cuando implementa el componente núcleo de Greengrass. Para obtener más información, consulte el parámetro de anulación de plataforma del componente núcleo de Greengrass.

Puede especificar uno de los siguientes valores para cada par clave-valor:

- Un valor exacto, como `linux` o `windows`. Los valores exactos deben comenzar por una letra o un número.
- `*`, que coincide con cualquier valor. Esto también coincide cuando un valor no está presente.
- Una expresión regular de estilo Java, como `/windows|linux/`. La expresión regular debe empezar y terminar con un carácter de barra inclinada (`/`). Por ejemplo, la expresión regular `/.+/` coincide con cualquier valor que no esté en blanco.

Este objeto contiene la siguiente información:

`runtime`

El tiempo de ejecución del núcleo de Greengrass para la plataforma compatible con este manifiesto. Al definir varios manifiestos con la plataforma `runtime`, los valores del tiempo de ejecución compatibles con una receta serán `aws_nucleus_lite` y `*` únicamente. Para dirigirse a un dispositivo clásico, el campo de tiempo de ejecución NO DEBE especificarse en la receta. Los tiempos de ejecución del núcleo de Greengrass compatibles incluyen los siguientes valores:

- `*`
- `aws_nucleus_lite`

`os`

(Opcional) El nombre del sistema operativo de la plataforma compatible con este manifiesto. Las plataformas comunes incluyen los siguientes valores:

- `linux`
- `windows`
- `darwin` (macOS)

`architecture`

(Opcional) La arquitectura de procesador de la plataforma compatible con este manifiesto. Las arquitecturas comunes incluyen los siguientes valores:

- `amd64`
- `arm`
- `aarch64`
- `x86`

`architecture.detail`

(Opcional) El detalle de la arquitectura del procesador de la plataforma compatible con este manifiesto. Los detalles de la arquitectura común incluyen los siguientes valores:

- `arm61`
- `arm71`
- `arm81`

key

(Opcional) Un atributo de plataforma que defina para este manifiesto. *Key*Sustitúyalo por el nombre del atributo de la plataforma. El software AWS IoT Greengrass Core hace coincidir este atributo de plataforma con los pares clave-valor que especifique en la configuración del componente núcleo de Greengrass. Para obtener más información, consulte el parámetro de anulación de plataforma del componente núcleo de Greengrass.

> **Tip**
>
> Use el formato de nombre de dominio inverso para evitar colisiones de nombres dentro de su empresa. Por ejemplo, si su empresa es propietaria de `example.com` y usted trabaja en un proyecto de radio, puede asignar un nombre a un atributo de plataforma personalizado `com.example.radio.RadioModule`. Esto ayuda a evitar colisiones entre los nombres de los atributos de la plataforma en su empresa.

Por ejemplo, puede definir un atributo de plataforma, `com.example.radio.RadioModule`, para especificar un manifiesto diferente en función del módulo de radio que esté disponible en un dispositivo principal. Cada manifiesto puede incluir diferentes artefactos que se apliquen a distintas configuraciones de hardware, de modo que se pueda implementar el conjunto mínimo de software en el dispositivo principal.

`Lifecycle`

Un objeto o cadena que define cómo instalar y ejecutar el componente en la plataforma que define este manifiesto. También puede definir un ciclo de vida global que se aplique a todas las plataformas. El dispositivo principal usa el ciclo de vida global solo si el manifiesto que se va a usar no especifica un ciclo de vida.

> **Note**
>
> Este ciclo de vida se define en un manifiesto. Los pasos del ciclo de vida que especifique aquí se aplican únicamente a la plataforma que define este manifiesto. También puede definir un ciclo de vida global que se aplique a todas las plataformas.

Este objeto o cadena contiene la siguiente información:

`Setenv`

(Opcional) Un diccionario de variables de entorno para proporcionarlo a todos los scripts del ciclo de vida. Puede anular estas variables de entorno con `Setenv` en cada script de ciclo de vida.

`install`

(Opcional) Un objeto o una cadena que define el script que se ejecutará cuando se instale el componente. El software AWS IoT Greengrass Core también ejecuta este paso del ciclo de vida cada vez que se lanza el software.

Si el script de `install` se cierra con un código de éxito, el componente entra en el estado `INSTALLED`.

Este objeto o cadena contiene la siguiente información:

`Script`

El script a ejecutar.

`RequiresPrivilege`

(Opcional) Puede ejecutar el script con privilegios de raíz. Si establece esta opción en`true`, el software AWS IoT Greengrass Core ejecutará este script de ciclo de vida como root en lugar de como el usuario del sistema que usted configure para ejecutar este componente. El valor predeterminado es `false`.

`Skipif`

(Opcional) La comprobación para determinar si se debe ejecutar o no el script. Puede definir que desea comprobar si hay un ejecutable en la ruta o si existe un archivo. Si el resultado es verdadero, el software AWS IoT Greengrass Core omite este paso. Elija una de las siguientes comprobaciones:

- `onpath` *`runnable`*: comprueba si hay un ejecutable en la ruta del sistema. Por ejemplo, use **`onpath python3`** para omitir este paso del ciclo de vida si Python 3 está disponible.
- `exists` *`file`*: comprueba si existe un archivo. Por ejemplo, use **`exists /tmp/my-configuration.db`** para omitir este paso del ciclo de vida si `/tmp/my-configuration.db` está presente.

`Timeout`

(Opcional) La cantidad máxima de tiempo en segundos que el script puede ejecutar antes de que el software AWS IoT Greengrass Core termine el proceso.

Valor predeterminado: 120 segundos

`Setenv`

(Opcional) El diccionario de variables de entorno que se va a proporcionar al script. Estas variables de entorno anulan las variables que usted proporciona en `Lifecycle.Setenv`.

`run`

(Opcional) Un objeto o una cadena que define el script que se ejecutará cuando se inicie el componente.

El componente entra en el estado `RUNNING` cuando se ejecuta este paso del ciclo de vida. Si el script de `run` se cierra con un código de éxito, el componente entra en el estado `STOPPING`. Si se especifica un script `shutdown`, se ejecuta; de lo contrario, el componente entra en estado `FINISHED`.

Los componentes que dependen de este componente se inician cuando se ejecuta este paso del ciclo de vida. Para ejecutar un proceso en segundo plano, como un servicio que usa los componentes dependientes, utilice el paso del ciclo de vida `startup` en su lugar.

Al implementar componentes con un ciclo de vida `run`, el dispositivo principal puede declarar que la implementación se ha completado en cuanto se ejecute este script de ciclo de vida. Como resultado, la implementación puede completarse y realizarse correctamente incluso si el script del ciclo de vida `run` falla poco después de ejecutarse. Si desea que el estado de la implementación dependa del resultado del script de inicio del componente, use el paso del ciclo de vida `startup` en su lugar.

> **Note**
>
> Puede definir solo un ciclo de vida, `startup` o `run`.

Este objeto o cadena contiene la siguiente información:

`Script`

El script a ejecutar.

`RequiresPrivilege`

(Opcional) Puede ejecutar el script con privilegios de raíz. Si establece esta opción en`true`, el software AWS IoT Greengrass Core ejecutará este script de ciclo de vida como root en lugar de como el usuario del sistema que usted configure para ejecutar este componente. El valor predeterminado es `false`.

`Skipif`

(Opcional) La comprobación para determinar si se debe ejecutar o no el script. Puede definir que desea comprobar si hay un ejecutable en la ruta o si existe un archivo. Si el resultado es verdadero, el software AWS IoT Greengrass Core omite el paso. Elija una de las siguientes comprobaciones:

- `onpath` *`runnable`*: comprueba si hay un ejecutable en la ruta del sistema. Por ejemplo, use **`onpath python3`** para omitir este paso del ciclo de vida si Python 3 está disponible.
- `exists` *`file`*: comprueba si existe un archivo. Por ejemplo, use **`exists /tmp/my-configuration.db`** para omitir este paso del ciclo de vida si `/tmp/my-configuration.db` está presente.

`Timeout`

(Opcional) El tiempo máximo en segundos que puede ejecutarse el script antes de que el software AWS IoT Greengrass principal finalice el proceso.

Este paso del ciclo de vida no agota el tiempo de espera de forma predeterminada. Si omite este tiempo de espera, el script `run` se ejecutará hasta que salga.

`Setenv`

(Opcional) El diccionario de variables de entorno que se va a proporcionar al script. Estas variables de entorno anulan las variables que usted proporciona en `Lifecycle.Setenv`.

`startup`

(Opcional) Un objeto o cadena que define el proceso en segundo plano que se ejecutará cuando se inicie el componente.

Use `startup` para ejecutar un comando que debe salir correctamente o para actualizar el estado del componente a `RUNNING` antes de que puedan iniciarse los componentes dependientes. Utilice la operación UpdateStateIPC para establecer el estado del componente como `RUNNING` o `ERRORED` cuando el componente inicie un script que no se cierre. Por ejemplo, puede definir un paso `startup` que inicie el proceso de MySQL con `/etc/init.d/mysqld start`.

El componente entra en el estado `STARTING` cuando se ejecuta este paso del ciclo de vida. Si el script de `startup` se cierra con un código de éxito, el componente entra en el estado `RUNNING`. A continuación, pueden iniciarse los componentes dependientes.

Al implementar componentes con un ciclo de vida `startup`, el dispositivo principal puede declarar que la implementación se ha completado una vez que este script de ciclo de vida sale o informa de su estado. En otras palabras, el estado de la implementación es `IN_PROGRESS` hasta que los scripts de startup de todos los componentes salgan o informen un estado.

> **Note**
>
> Puede definir solo un ciclo de vida, `startup` o `run`.

Este objeto o cadena contiene la siguiente información:

`Script`

El script a ejecutar.

`RequiresPrivilege`

(Opcional) Puede ejecutar el script con privilegios de raíz. Si establece esta opción en `true`, el software AWS IoT Greengrass principal ejecutará este script de ciclo de vida como root en lugar de como el usuario del sistema que haya configurado para ejecutar este componente. El valor predeterminado es `false`.

`Skipif`

(Opcional) La comprobación para determinar si se debe ejecutar o no el script. Puede definir que desea comprobar si hay un ejecutable en la ruta o si existe un archivo. Si el resultado es verdadero, el software AWS IoT Greengrass Core omite el paso. Elija una de las siguientes comprobaciones:

- `onpath` *`runnable`*: comprueba si hay un ejecutable en la ruta del sistema. Por ejemplo, use **`onpath python3`** para omitir este paso del ciclo de vida si Python 3 está disponible.
- `exists` *`file`*: comprueba si existe un archivo. Por ejemplo, use **`exists /tmp/my-configuration.db`** para omitir este paso del ciclo de vida si `/tmp/my-configuration.db` está presente.

`Timeout`

(Opcional) La cantidad máxima de tiempo en segundos que el script puede ejecutar antes de que el software AWS IoT Greengrass Core termine el proceso.

Valor predeterminado: 120 segundos

`Setenv`

(Opcional) El diccionario de variables de entorno que se va a proporcionar al script. Estas variables de entorno anulan las variables que usted proporciona en `Lifecycle.Setenv`.

`shutdown`

(Opcional) Un objeto o una cadena que define el script que se ejecutará cuando el componente se apague. Use el ciclo de vida de apagado para ejecutar el código que desee ejecutar cuando el componente esté en el estado `STOPPING`. El ciclo de vida de apagado se puede usar para detener un proceso iniciado por el `startup` o por los scripts `run`.

Si inicia un proceso en segundo plano en `startup`, use el paso `shutdown` para detener ese proceso cuando el componente se apague. Por ejemplo, puede definir un paso `shutdown` que detenga el proceso de MySQL con `/etc/init.d/mysqld stop`.

El script `shutdown` se ejecuta después de que el componente entre en el estado `STOPPING`. Si el script se completa de forma satisfactoria, el componente entra en el estado `FINISHED`.

Este objeto o cadena contiene la siguiente información:

`Script`

El script a ejecutar.

`RequiresPrivilege`

(Opcional) Puede ejecutar el script con privilegios de raíz. Si establece esta opción en`true`, el software AWS IoT Greengrass principal ejecutará este script de ciclo de vida como root en lugar de como el usuario del sistema que usted configure para ejecutar este componente. El valor predeterminado es `false`.

`Skipif`

(Opcional) La comprobación para determinar si se debe ejecutar o no el script. Puede definir que desea comprobar si hay un ejecutable en la ruta o si existe un archivo. Si el resultado es verdadero, el software AWS IoT Greengrass Core omite el paso. Elija una de las siguientes comprobaciones:

- `onpath` *`runnable`*: comprueba si hay un ejecutable en la ruta del sistema. Por ejemplo, use **`onpath python3`** para omitir este paso del ciclo de vida si Python 3 está disponible.
- `exists` *`file`*: comprueba si existe un archivo. Por ejemplo, use **`exists /tmp/my-configuration.db`** para omitir este paso del ciclo de vida si `/tmp/my-configuration.db` está presente.

`Timeout`

(Opcional) El tiempo máximo en segundos que puede ejecutarse el script antes de que el software AWS IoT Greengrass principal finalice el proceso.

Valor predeterminado: 15 segundos

`Setenv`

(Opcional) El diccionario de variables de entorno que se va a proporcionar al script. Estas variables de entorno anulan las variables que usted proporciona en `Lifecycle.Setenv`.

`recover`

(Opcional) Un objeto o una cadena que define el script que se ejecutará cuando el componente detecte un error.

Este paso se ejecuta cuando un componente entra en el estado `ERRORED`. Si el componente pasa a `ERRORED` tres veces sin recuperarse correctamente, el componente cambia al estado `BROKEN`. Para corregir un componente `BROKEN`, debe volver a implementarlo.

Este objeto o cadena contiene la siguiente información:

`Script`

El script a ejecutar.

`RequiresPrivilege`

(Opcional) Puede ejecutar el script con privilegios de raíz. Si establece esta opción en`true`, el software AWS IoT Greengrass principal ejecutará este script de ciclo de vida como root en lugar de como el usuario del sistema que haya configurado para ejecutar este componente. El valor predeterminado es `false`.

`Skipif`

(Opcional) La comprobación para determinar si se debe ejecutar o no el script. Puede definir que desea comprobar si hay un ejecutable en la ruta o si existe un archivo. Si el resultado es verdadero, el software AWS IoT Greengrass Core omite el paso. Elija una de las siguientes comprobaciones:

- `onpath` *`runnable`*: comprueba si hay un ejecutable en la ruta del sistema. Por ejemplo, use **`onpath python3`** para omitir este paso del ciclo de vida si Python 3 está disponible.
- `exists` *`file`*: comprueba si existe un archivo. Por ejemplo, use **`exists /tmp/my-configuration.db`** para omitir este paso del ciclo de vida si `/tmp/my-configuration.db` está presente.

`Timeout`

(Opcional) El tiempo máximo en segundos que puede ejecutarse el script antes de que el software AWS IoT Greengrass principal finalice el proceso.

Valor predeterminado: 60 segundos.

`Setenv`

(Opcional) El diccionario de variables de entorno que se va a proporcionar al script. Estas variables de entorno anulan las variables que usted proporciona en `Lifecycle.Setenv`.

`bootstrap`

(Opcional) Un objeto o una cadena que define un script que requiere el reinicio del software AWS IoT Greengrass Core o del dispositivo principal. Esto le permite desarrollar

un componente que se reinicie después de instalar las actualizaciones del sistema operativo o las actualizaciones del tiempo de ejecución, por ejemplo.

> **Note**
>
> Para instalar actualizaciones o dependencias que no requieran el reinicio del software o dispositivo AWS IoT Greengrass principal, usa el ciclo de vida de instalación.

Este paso del ciclo de vida se ejecuta antes del paso del ciclo de vida de la instalación en los siguientes casos cuando el software AWS IoT Greengrass principal implementa el componente:

- El componente se implementa en el dispositivo principal por primera vez.
- La versión del componente cambia.
- El script de arranque cambia como resultado de una actualización de la configuración del componente.

Una vez que el software AWS IoT Greengrass principal complete el paso de arranque de todos los componentes que tengan un paso de arranque en una implementación, el software se reinicia.

> **Important**
>
> Debe configurar el software AWS IoT Greengrass Core como un servicio del sistema para reiniciar el software AWS IoT Greengrass Core o el dispositivo principal. Si no configura el software AWS IoT Greengrass principal como un servicio del sistema, el software no se reiniciará. Para obtener más información, consulte Configuración del núcleo de Greengrass como un servicio del sistema.

Este objeto o cadena contiene la siguiente información:

`BootstrapOnRollback`

> **Note**
>
> Cuando esta característica está habilitada, solo `BootstrapOnRollback` se ejecutará en los componentes que hayan completado o intentado ejecutar los

pasos del ciclo de vida del arranque como parte de una implementación de destino fallida. Esta característica está disponible para las versiones 2.12.0 y posteriores del núcleo de Greengrass.

(Opcional) Puede ejecutar los pasos del ciclo de vida de arranque como parte de una implementación de reversión. Si configura esta opción en `true`, se ejecutarán los pasos del ciclo de vida de arranque definidos en una implementación de reversión. Cuando se produce un error en una implementación, la versión anterior del ciclo de vida de arranque del componente se volverá a ejecutar durante una implementación de reversión.

El valor predeterminado es `false`.

`Script`

El script a ejecutar. El código de salida de este script define la instrucción de reinicio. Use los siguientes códigos de salida:

- `0`— No reinicies el software AWS IoT Greengrass principal ni el dispositivo principal. El software AWS IoT Greengrass principal se sigue reiniciando después del arranque de todos los componentes.
- `100`— Solicitud para reiniciar el software AWS IoT Greengrass Core.
- `101`: solicitud para reiniciar el dispositivo principal.

Los códigos de salida 100 a 199 están reservados para un comportamiento especial. Otros códigos de salida representan errores de script.

`RequiresPrivilege`

(Opcional) Puede ejecutar el script con privilegios de raíz. Si establece esta opción en`true`, el software AWS IoT Greengrass principal ejecutará este script de ciclo de vida como root en lugar de como el usuario del sistema que usted configure para ejecutar este componente. El valor predeterminado es `false`.

`Timeout`

(Opcional) La cantidad máxima de tiempo en segundos que el script puede ejecutar antes de que el software AWS IoT Greengrass Core termine el proceso.

Valor predeterminado: 120 segundos

`Setenv`

(Opcional) El diccionario de variables de entorno que se va a proporcionar al script. Estas variables de entorno anulan las variables que usted proporciona en `Lifecycle.Setenv`.

`Selections`

(Opcional) Una lista de claves de selección que especifican las secciones del ciclo de vida global que se van a ejecutar en este manifiesto. En el ciclo de vida global, puede definir los pasos del ciclo de vida con claves de selección en cualquier nivel para seleccionar subsecciones del ciclo de vida. A continuación, el dispositivo principal usa las secciones que coinciden con las claves de selección de este manifiesto. Para obtener más información, consulte los ejemplos del ciclo de vida global.

> **Important**
>
> El dispositivo principal usa las selecciones del ciclo de vida global solo si este manifiesto no define un ciclo de vida.

Puede especificar la clave de selección `all` para ejecutar secciones del ciclo de vida global que no tienen claves de selección.

`Artifacts`

(Opcional) Una lista de objetos que definen cada uno un artefacto binario para el componente de la plataforma que define este manifiesto. Por ejemplo, puede definir código o imágenes como artefactos.

Cuando el componente se implementa, el software AWS IoT Greengrass principal descarga el artefacto en una carpeta del dispositivo principal. También puede definir los artefactos como archivos de almacenamiento que el software extrae después de descargarlos.

Puede usar variables de receta para obtener las rutas a las carpetas en las que se instalan los artefactos en el dispositivo principal.

- Archivos normales: use la variable de receta artifacts:path para obtener la ruta a la carpeta que contiene los artefactos. Por ejemplo, especifique `{artifacts:path}/my_script.py` en una receta para obtener la ruta a un artefacto que tenga el URI `s3://amzn-s3-demo-bucket/path/to/my_script.py`.

- Archivos extraídos: use la variable de receta artifacts:decompressedPath para obtener la ruta a la carpeta que contiene los artefactos de archivo extraídos. El software AWS IoT Greengrass Core extrae cada archivo a una carpeta con el mismo nombre que el archivo. Por ejemplo, especifique `{artifacts:decompressedPath}/my_archive/my_script.py` en una receta para obtener la ruta a `my_script.py` en el artefacto de archivo que contiene el URI `s3://amzn-s3-demo-bucket/path/to/my_archive.zip`.

> **Note**
>
> Al desarrollar un componente con un artefacto de archivo en un dispositivo principal local, es posible que no disponga de un URI para ese artefacto. Para probar el componente con una opción `Unarchive` que extraiga el artefacto, especifique un URI en el que el nombre del archivo coincida con el nombre del archivo del artefacto archivado. Puede especificar el URI en el que desea cargar el artefacto de archivo o puede especificar un nuevo URI de marcador de posición. Por ejemplo, para extraer el artefacto `my_archive.zip` durante una implementación local, puede especificar `s3://amzn-s3-demo-bucket/my_archive.zip`.

Cada objeto contiene la siguiente información:

`Uri`

El URI de un artefacto en un bucket de S3. El software AWS IoT Greengrass Core obtiene el artefacto de este URI cuando se instala el componente, a menos que el artefacto ya exista en el dispositivo. Cada artefacto debe tener un nombre de archivo único en cada manifiesto.

`Unarchive`

(Opcional) El tipo de archivo que se va a desempaquetar. Puede elegir entre las siguientes opciones:

- `NONE`: el archivo no es un archivo para desempaquetar. El software AWS IoT Greengrass Core instala el artefacto en una carpeta del dispositivo principal. Puede usar la variable de receta artifacts:path para obtener la ruta a esta carpeta.
- `ZIP`: el archivo es un archivo ZIP. El software AWS IoT Greengrass Core extrae el archivo a una carpeta con el mismo nombre que el archivo. Puede usar la variable de

receta artifacts:decompressedPath para obtener la ruta a la carpeta que contiene esta carpeta.

El valor predeterminado es `NONE`.

`Permission`

(Opcional) Un objeto que define los permisos de acceso que se van a establecer para este archivo de artefacto. Puede establecer el permiso de lectura y el permiso de ejecución.

> **Note**
>
> No puede configurar el permiso de escritura porque el software AWS IoT Greengrass Core no permite que los componentes editen los archivos de artefactos de la carpeta de artefactos. Para editar un archivo de artefactos de un componente, cópielo en otra ubicación o publique e implemente un nuevo archivo de artefacto.

Si defines un artefacto como un archivo para desempaquetar, el software AWS IoT Greengrass Core establece estos permisos de acceso en los archivos que desempaqueta del archivo. El software AWS IoT Greengrass Core establece los permisos de acceso de la carpeta en para `ALL` y. `Read Execute` Esto permite a los componentes ver los archivos desempaquetados de la carpeta. Para establecer los permisos en los archivos individuales del archivo, puede configurarlos en el script del ciclo de vida de instalación.

Este objeto contiene la siguiente información:

`Read`

(Opcional) El permiso de lectura que se debe configurar para este archivo de artefacto. Para permitir que otros componentes accedan a este artefacto, como los componentes que dependen de este componente, especifique `ALL`. Puede elegir entre las siguientes opciones:

- `NONE`: el archivo no se puede leer.
- `OWNER`: el archivo se puede leer por el usuario del sistema que haya configurado para ejecutar este componente.
- `ALL`: el archivo se puede leer por todos los usuarios.

El valor predeterminado es `OWNER`.

`Execute`

(Opcional) El permiso de ejecución que se debe configurar para este archivo de artefacto. El permiso `Execute` implica el permiso `Read`. Por ejemplo, si especifica `ALL` para `Execute`, todos los usuarios podrán leer y ejecutar este archivo de artefacto.

Puede elegir entre las siguientes opciones:

- `NONE`: el archivo no se puede ejecutar.
- `OWNER`: el archivo se puede ejecutar por el usuario del sistema que usted configure para ejecutar el componente.
- `ALL`: el archivo se puede ejecutar por todos los usuarios.

El valor predeterminado es `NONE`.

`Digest`

(Solo lectura) El hash criptográfico resumido del artefacto. Al crear un componente, AWS IoT Greengrass utiliza un algoritmo de hash para calcular el hash del archivo del artefacto. A continuación, al implementar el componente, el núcleo de Greengrass calcula el hash del artefacto descargado y lo compara con este resumen para verificar el artefacto antes de la instalación. Si el hash no coincide con el resumen, se produce un error en la implementación.

Si establece este parámetro, AWS IoT Greengrass reemplaza el valor que estableció al crear el componente.

`Algorithm`

(Solo lectura) El algoritmo de hash que se AWS IoT Greengrass utiliza para calcular el hash resumido del artefacto.

Si establece este parámetro, AWS IoT Greengrass reemplaza el valor que estableció al crear el componente.

`Lifecycle`

Un objeto que define cómo instalar y ejecutar el componente. El dispositivo principal usa el ciclo de vida global solo si el manifiesto que se va a usar no especifica un ciclo de vida.

> **Note**
>
> Este ciclo de vida se define fuera de un manifiesto. También puede definir un ciclo de vida del manifiesto que se aplique a las plataformas que coincidan con ese manifiesto.

En el ciclo de vida global, puede especificar los ciclos de vida que se ejecutan para determinadas claves de selección que especifica en cada manifiesto. Las claves de selección son cadenas que identifican las secciones del ciclo de vida global que se van a ejecutar en cada manifiesto.

La clave de selección `all` es la predeterminada en cualquier sección sin una clave de selección. Esto significa que puede especificar la clave de selección `all` en un manifiesto para ejecutar las secciones del ciclo de vida global sin claves de selección. No es necesario especificar la clave de selección `all` en el ciclo de vida global.

Si un manifiesto no define un ciclo de vida o claves de selección, el dispositivo principal usará la selección `all` de forma predeterminada. Esto significa que, en este caso, el dispositivo principal usa las secciones del ciclo de vida global que no usan claves de selección.

Este objeto contiene la misma información que el ciclo de vida del manifiesto, pero puede especificar las claves de selección en cualquier nivel para seleccionar subsecciones del ciclo de vida.

> **Tip**
>
> Le recomendamos que use solo letras minúsculas para cada clave de selección para evitar conflictos entre las claves de selección y las claves del ciclo de vida. Las claves del ciclo de vida comienzan por una letra mayúscula.

Example Ejemplo de ciclo de vida global con claves de selección de nivel superior

```
Lifecycle:
  key1:
    install:
      SkipIf: either onpath executable or exists file
      Script: command1
  key2:
    install:
```

```
      Script: command2
  all:
    install:
      Script: command3
```

Example Ejemplo de ciclo de vida global con claves de selección de nivel inferior

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Example Ejemplo de ciclo de vida global con claves de selección de varios niveles

```
Lifecycle:
  key1:
    install:
      SkipIf: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script:
        key3: command3
        key4: command4
        all: command5
```

Variables de receta

Las variables de receta exponen la información del componente y el núcleo actuales para que la use en sus recetas. Por ejemplo, puede usar una variable de receta para pasar los parámetros de configuración de los componentes a una aplicación que ejecute en un script de ciclo de vida.

Puede usar variables de receta en las siguientes secciones de recetas de componentes:

- Definiciones del ciclo de vida.

- Definiciones de configuración de componentes, si utiliza Greengrass nucleus v2.6.0 o posterior y establece la interpolateComponentConfigurationopción de configuración en. `true` También puede usar variables de receta al implementar las actualizaciones de configuración de los componentes.

Las variables de receta usan la sintaxis `{recipe_variable}`. Los corchetes indican una variable de receta.

AWS IoT Greengrass admite las siguientes variables de receta:

`component_dependency_name``:configuration:`*`json_pointer`*

El valor de un parámetro de configuración para el componente que define esta receta o para un componente del que depende este componente.

Puede usar esta variable para proporcionar un parámetro a un script que ejecute en el ciclo de vida del componente.

> **Note**
>
> AWS IoT Greengrass admite esta variable de receta solo en las definiciones del ciclo de vida de los componentes.

Esta variable de receta tiene las siguientes entradas:

- `component_dependency_name`: (opcional) el nombre de la dependencia del componente que se consultará. Omita este segmento para consultar el componente que define esta receta. Puede especificar solo las dependencias directas.
- `json_pointer`: el puntero JSON al valor de configuración que se va a evaluar. Los punteros JSON comienzan con una barra diagonal `/`. Para identificar un valor en una configuración de componentes anidados, use barras diagonales (`/`) para separar las claves de cada nivel de la configuración. Puede usar un número como clave para especificar un índice en una lista. Para obtener más información, consulte la especificación de puntero JSON.

  AWS IoT Greengrass Core usa punteros JSON para recetas en formato YAML.

El puntero JSON puede hacer referencia a los siguientes tipos de nodos:

- Un nodo de valor. AWS IoT Greengrass Core reemplaza la variable de receta por la representación en cadena del valor. Los valores nulos se convierten a `null` como una cadena.

- Un nodo de objeto. AWS IoT Greengrass Core reemplaza la variable de receta por la representación de cadena JSON serializada de ese objeto.
- Sin nodo. AWS IoT Greengrass Core no reemplaza la variable de receta.

Por ejemplo, la variable de receta `{configuration:/Message}` recupera el valor de la clave `Message` en la configuración del componente. La variable de receta `{com.example.MyComponentDependency:configuration:/server/port}` recupera el valor de `port` en el objeto de configuración `server` de una dependencia de un componente.

`component_dependency_name``:artifacts:path`

La ruta raíz de los artefactos del componente que define esta receta o de un componente del que depende este componente.

Cuando se instala un componente, AWS IoT Greengrass copia los artefactos del componente en la carpeta que expone esta variable. Puede usar esta variable para identificar la ubicación de un script que se va a ejecutar en el ciclo de vida del componente, por ejemplo.

La carpeta de esta ruta es de solo lectura. Para modificar los archivos de artefactos, cópielos en otra ubicación, como el directorio de trabajo actual (`$PWD` o `.`). A continuación, modifique los archivos allí.

Para leer o ejecutar un artefacto desde una dependencia de un componente, el permiso del artefacto `Read` o `Execute` debe ser `ALL`. Para obtener más información, consulta los permisos de artefactos que define en la receta del componente.

Esta variable de receta tiene las siguientes entradas:

- `component_dependency_name`: (opcional) el nombre de la dependencia del componente que se consultará. Omita este segmento para consultar el componente que define esta receta. Puede especificar solo las dependencias directas.

`component_dependency_name``:artifacts:decompressedPath`

La ruta raíz de los artefactos de archivo descomprimidos para el componente que define esta receta o para un componente del que depende este componente.

Cuando se instala un componente, AWS IoT Greengrass desempaqueta los artefactos archivados del componente en la carpeta que expone esta variable. Puede usar esta variable para identificar la ubicación de un script que se va a ejecutar en el ciclo de vida del componente, por ejemplo.

Cada artefacto se descomprime en una carpeta dentro de la ruta descomprimida, donde la carpeta tiene el mismo nombre que el artefacto menos su extensión. Por

ejemplo, un artefacto ZIP denominado `models.zip` se descomprime en la carpeta `{artifacts:decompressedPath}/models`.

La carpeta de esta ruta es de solo lectura. Para modificar los archivos de artefactos, cópielos en otra ubicación, como el directorio de trabajo actual (`$PWD` o `.`). A continuación, modifique los archivos allí.

Para leer o ejecutar un artefacto desde una dependencia de un componente, el permiso del artefacto `Read` o `Execute` debe ser `ALL`. Para obtener más información, consulta los permisos de artefactos que define en la receta del componente.

Esta variable de receta tiene las siguientes entradas:

- `component_dependency_name`: (opcional) el nombre de la dependencia del componente que se consultará. Omita este segmento para consultar el componente que define esta receta. Puede especificar solo las dependencias directas.

`component_dependency_name``:work:path`

Esta característica está disponible para la versión 2.0.4 y versiones posteriores del componente núcleo de Greengrass.

La ruta de trabajo del componente que define esta receta o de un componente del que depende este componente. El valor de esta variable de receta equivale a la salida de la variable de entorno `$PWD` y del comando pwd cuando se ejecuta desde el contexto del componente.

Puede usar esta variable de receta para compartir archivos entre un componente y una dependencia.

El componente que define esta receta y los demás componentes que se ejecutan como el mismo usuario y grupo pueden leer y escribir en la carpeta de esta ruta.

Esta variable de receta tiene las siguientes entradas:

- `component_dependency_name`: (opcional) el nombre de la dependencia del componente que se consultará. Omita este segmento para consultar el componente que define esta receta. Puede especificar solo las dependencias directas.

`kernel:rootPath`

La ruta raíz AWS IoT Greengrass principal.

`iot:thingName`

Esta característica está disponible para la versión 2.3.0 y versiones posteriores del componente núcleo de Greengrass.

El nombre del dispositivo AWS IoT principal.

Ejemplos de receta

Puede hacer referencia a los siguientes ejemplos de recetas para ayudarlo a crear recetas para sus componentes.

AWS IoT Greengrass selecciona un índice de componentes de Greengrass, denominado Catálogo de software de Greengrass. Este catálogo rastrea los componentes de Greengrass desarrollados por la comunidad de Greengrass. Desde este catálogo, puede descargar, modificar e implementar componentes para crear sus aplicaciones de Greengrass. Para obtener más información, consulte Componentes de la comunidad.

Temas

- Receta de componentes Hello World
- Ejemplo de componente de tiempo de ejecución de Python
- Receta de componente que especifica varios campos

Receta de componentes Hello World

En la siguiente receta, se describe un componente Hello World que ejecuta un script de Python. Este componente es compatible con todas las plataformas y acepta un parámetro `Message` que AWS IoT Greengrass pasa como argumento al script de Python. Esta es la receta del componente Hello World del tutorial de introducción.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
```

```
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
```

```
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Ejemplo de componente de tiempo de ejecución de Python

La siguiente receta describe un componente que instala Python. Este componente es compatible con dispositivos Linux de 64 bits.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": "apt-get update\napt-get install python3.7"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
  - Platform:
      os: linux
      architecture: amd64
```

```
    Lifecycle:
      install: |
        apt-get update
        apt-get install python3.7
```

Receta de componente que especifica varios campos

La siguiente receta de componentes usa varios campos de receta.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
  },
  "ComponentDependencies": {
    "BarService": {
      "VersionRequirement": "^1.1.0",
      "DependencyType": "SOFT"
    },
    "BazService": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git"
        },
        "Setenv": {
```

```
          "environment_variable1": "variable_value1",
          "environment_variable2": "variable_value2"
        }
      },
      "Artifacts": [
        {
          "Uri": "s3://amzn-s3-demo-bucket/hello_world.zip",
          "Unarchive": "ZIP"
        },
        {
          "Uri": "s3://amzn-s3-demo-bucket/hello_world_linux.py"
        }
      ]
    },
    {
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git",
          "RequiresPrivilege": "true"
        }
      },
      "Artifacts": [
        {
          "Uri": "s3://amzn-s3-demo-bucket/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
```

```
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        SkipIf: onpath git
        Script: sudo apt-get install git
      SetEnv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - Uri: 's3://amzn-s3-demo-bucket/hello_world.zip'
        Unarchive: ZIP
      - Uri: 's3://amzn-s3-demo-bucket/hello_world_linux.py'
  - Lifecycle:
      install:
        SkipIf: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - Uri: 's3://amzn-s3-demo-bucket/hello_world.py'
```

Referencia de variable de entorno del componente

El software AWS IoT Greengrass principal establece las variables de entorno cuando ejecuta los scripts del ciclo de vida de los componentes. Puede incluir estas variables de entorno en sus componentes para obtener el nombre de la cosa y la Región de AWS versión del núcleo de Greengrass. El software también establece las variables de entorno que su componente necesita para usar el SDK de comunicación entre procesos y para interactuar con los servicios de AWS.

También puede configurar variables de entorno personalizadas para los scripts del ciclo de vida de su componente. Para obtener más información, consulte Setenv.

El software AWS IoT Greengrass Core establece las siguientes variables de entorno:

`AWS_IOT_THING_NAME`

El nombre de AWS IoT lo que representa este dispositivo central de Greengrass.

`AWS_REGION`

Es Región de AWS donde funciona este dispositivo central de Greengrass.

AWS SDKs Usan esta variable de entorno para identificar la región que se va a utilizar por defecto. Esta variable es equivalente a `AWS_DEFAULT_REGION`.

`AWS_DEFAULT_REGION`

Es Región de AWS donde funciona este dispositivo central de Greengrass.

AWS CLI Utiliza esta variable de entorno para identificar la región predeterminada que se va a utilizar. Esta variable es equivalente a `AWS_REGION`.

`GGC_VERSION`

La versión del componente de núcleo de Greengrass que se ejecuta en este dispositivo principal de Greengrass.

`GG_ROOT_CA_PATH`

Esta característica está disponible para la versión 2.5.5 y versiones posteriores del componente de núcleo de Greengrass.

La ruta a la autoridad del certificado (CA) raíz certifica que usa el núcleo de Greengrass.

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

La ruta al socket IPC que utilizan los componentes para comunicarse con el software AWS IoT Greengrass Core. Para obtener más información, consulte Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core.

`SVCUID`

El token secreto que utilizan los componentes para conectarse al socket IPC y comunicarse con el software AWS IoT Greengrass Core. Para obtener más información, consulte Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core.

`AWS_CONTAINER_AUTHORIZATION_TOKEN`

El token secreto que usan los componentes para recuperar las credenciales del componente del servicio de intercambio de tokens.

`AWS_CONTAINER_CREDENTIALS_FULL_URI`

El URI que solicitan los componentes para recuperar las credenciales del componente del servicio de intercambio de tokens.

Implemente AWS IoT Greengrass componentes en los dispositivos

Puede utilizarlos AWS IoT Greengrass para implementar componentes en dispositivos o grupos de dispositivos. Las implementaciones se utilizan para definir los componentes y las configuraciones que se envían a los dispositivos. AWS IoT Greengrass se despliega en objetivos, AWS IoT cosas o grupos de cosas que representan los dispositivos principales de Greengrass. AWS IoT Greengrass utiliza AWS IoT Core tareas para implementarlas en sus dispositivos principales. Puede configurar la forma en que se implementará el trabajo en sus dispositivos.

Implementaciones en dispositivos principales

Cada dispositivo principal ejecuta los componentes de las implementaciones de ese dispositivo. Una nueva implementación en el mismo destino sobrescribe la implementación anterior en el destino. Cuando crea una implementación, define los componentes y las configuraciones que se van a aplicar al software existente del dispositivo principal.

Cuando revisa una implementación para un destino, sustituye los componentes de la revisión anterior por los componentes de la nueva revisión. Por ejemplo, implementa los componentes Administrador de registros y Administrador de secretos en el grupo de objetos `TestGroup`. A continuación, se crea otra implementación para `TestGroup` que especifica solo el componente administrador de secretos. Como resultado, los dispositivos principales de ese grupo ya no ejecutan el administrador de registros.

Resolución de dependencias de plataformas

Cuando un dispositivo principal recibe una implementación, comprueba que los componentes son compatibles con el dispositivo principal. Por ejemplo, si implementa el Firehose en un destino de Windows, la implementación fallará.

Resolución de dependencias de componentes

Durante la implementación de un componente, el dispositivo principal verifica la compatibilidad de todas las dependencias de los componentes y los requisitos de versión en un grupo de objetos. Esta

verificación garantiza que se cumplan las restricciones de versión de todos los componentes y sus dependencias antes de continuar con la implementación.

El proceso de resolución de dependencias comienza al identificar los componentes de destino que no tienen dependencias en sus recetas. Luego, el sistema construye un árbol de dependencias mediante la búsqueda en anchura (BFS), que explora sistemáticamente cada nodo de destino y encuentra primero sus dependencias antes de pasar al siguiente nodo. Cada nodo incluye el componente de destino como clave y los requisitos de versión como valor.

Los requisitos de versión combinan tres conjuntos de restricciones:

- Los requisitos de versión que ya están establecidos en el grupo de objetos existente.
- La versión del componente exigida por la implementación. Debe seleccionar una versión del componente al realizar una implementación o al actualizarla.
- Toda restricción de versión de un componente que esté definida en la sección de dependencias de la receta.

Resolución de las dependencias de componentes

En una implementación, el núcleo de Greengrass primero intenta encontrar el candidato local que se está ejecutando actualmente en el dispositivo que cumple los requisitos. Si el componente en ejecución cumple los requisitos, el núcleo obtiene la ruta de recetas almacenada en la carpeta de recetas y encuentra la versión local más reciente en el almacén local.

En el Nube de AWS caso de las implementaciones, el núcleo llamará entonces a la ResolveComponentCandidates API. Esta API comenzará con la versión más reciente disponible y comprobará si cumple las dependencias y los requisitos. El núcleo selecciona la versión más reciente cuando recibe la respuesta de la API. Si no se encuentra ninguna versión de la Nube de AWS que cumpla los requisitos, se producirá un error en la implementación. Si el dispositivo está desconectado, recurre al candidato local original encontrado. Pero si no se encuentra ningún candidato local que cumpla los requisitos, la implementación no se realizará correctamente.

Para los despliegues locales, el núcleo utiliza exclusivamente candidatos locales si existen y si cumplen los requisitos sin negociar. Nube de AWS Si no existe un candidato de este tipo, la implementación devolverá un error.

> **Note**
>
> Todas las recetas resueltas se almacenan de formal local para futuras consultas.

Para obtener más información, consulte la sección de resolución de dependencias en GitHub.

Si el núcleo de Greengrass es capaz de resolver correctamente todos los componentes, el registro del núcleo contendrá la siguiente línea.

```
resolve-all-group-dependencies-finish. Finish resolving all groups dependencies.
```

Example

El siguiente ejemplo muestra cómo el núcleo resuelve los requisitos de los componentes.

- Se implementa ComponentA, que depende de las versiones 1.0.0-1.9.0 del ComponentC.
- También se implementa ComponentB, que depende de las versiones 1.4.0-1.9.5 del ComponentC.

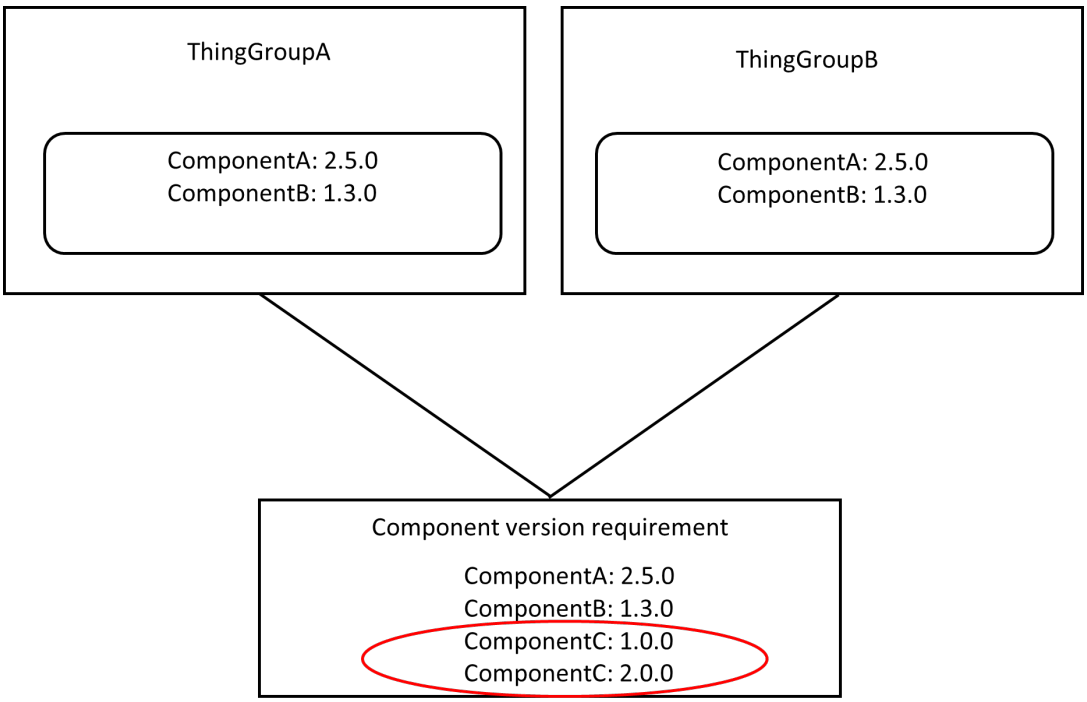
Con la resolución de dependencia de los componentes, el núcleo implementará la versión más reciente del ComponentC para cumplir con los requisitos del ComponentA y el ComponentB. La versión más reciente de ComponentC es 1.9.0.

Errores comunes en la resolución de dependencias de componentes

La resolución de dependencias de componentes puede fallar por dos motivos principales: un conflicto de requisitos de la versión de destino o un conflicto de requisitos de dependencia de componentes.

Situación 1: conflicto de requisitos de la versión de destino

- Existe un objeto en un grupo de objetos, y también desea agregarlo a un nuevo grupo de objetos. La implementación fallará si el nuevo grupo de objetos requiere una versión diferente.
- Una implementación también puede fallar si un objeto pertenece a un grupo de objetos y quiere actualizar la versión del componente mediante la implementación de un objeto.



Ejemplo de registro de errores:

```
2025-04-11T06:16:03.315Z [ERROR] (pool-3-thread-27)
 com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
 with cloud and no local version to fall back to. {componentName=ComponentC,
 versionRequirement={thing/ABC==2.0.0, thinggroup/ThingGroupA==1.0.0}}
2025-04-11T06:16:03.316Z [ERROR] (pool-3-thread-26)
 com.aws.greengrass.deployment.DeploymentService: Error occurred while
 processing deployment. {deploymentId=fbac24de-4ef9-44b0-a685-fdc63b0f02b8,
 serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentC version constraints:
 thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
```

```
    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentC version constraints:
 thing/ABC requires =2.0.0, thinggroup/ThingGroupA requires =1.0.0.
    at
 com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManage
    at
 com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)
    ... 4 more
```

Los registros muestran un error de conflicto de versiones, ya que el núcleo no puede encontrar una versión de un componente que cumpla simultáneamente dos requisitos contradictorios.

Cómo resolverlo

- Si desea mantener el componente en cada grupo de objetos, seleccione la misma versión de ese componente en cada grupo de objetos.
- Seleccione una versión del componente que cumpla con los requisitos de implementación.
- Si desea utilizar una versión del componente que no cumpla los requisitos de ambos grupos de objetos, seleccione la versión del componente que cumpla con el requisito de versión del grupo de objetos y utilícelo solo en ese grupo de objetos.

Situación 2: conflicto entre los requisitos de la versión de dependencia de los componentes

Si un componente es una dependencia de diferentes componentes y los componentes requieren diferentes versiones o rangos de versiones de ese componente, existe la posibilidad de que no haya versiones disponibles que satisfagan todos los requisitos de versión. En este caso, la implementación producirá un error.

Example

Implementación del ComponentA (versión 2.5.0), el ComponentB (versión 1.3.0) y ComponentC (versión 1.0.0)

- ComponentA requiere una versión mayor o igual a 1.0.0 de ComponentB

```
---
...
ComponentName: ComponentA
ComponentVersion: "2.5.0"
ComponentDependencies:
    ComponentB:
        VersionRequirement: ">=1.0.0"
        DependencyType: "HARD"
...
```

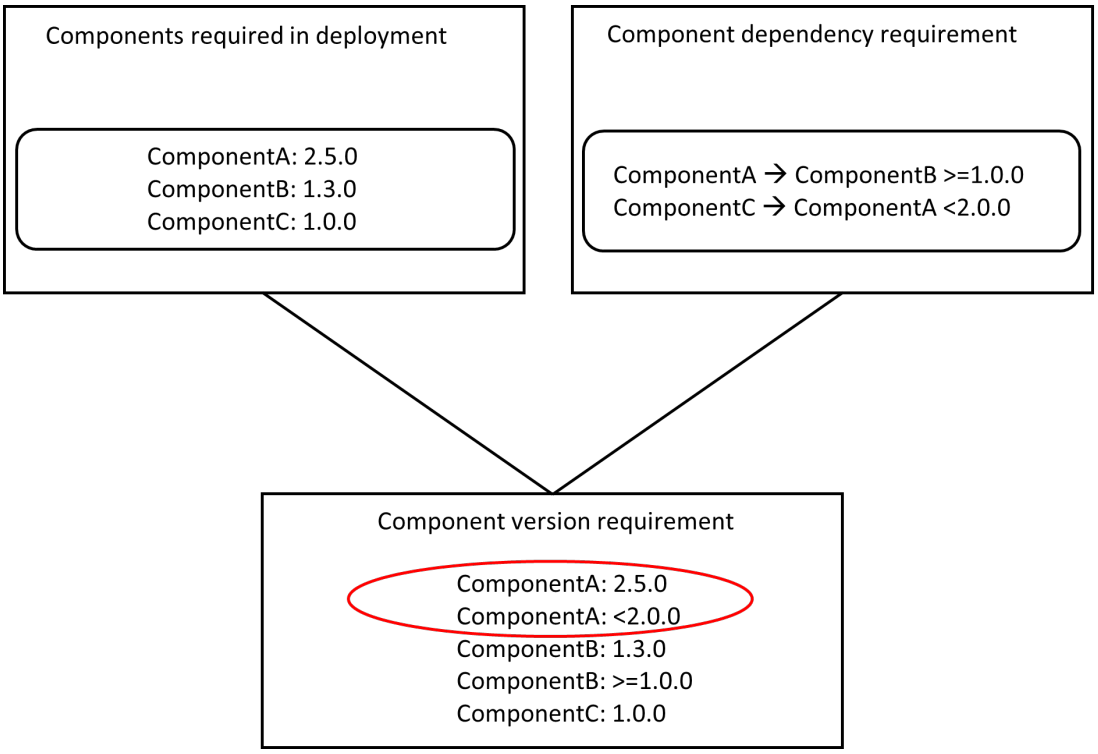
- ComponentC requiere una versión menor a 2.0.0 de ComponentA

```
---
...
ComponentName: ComponentC
ComponentVersion: "1.0.0"
ComponentDependencies:
    ComponentA:
        VersionRequirement: "<2.0.0"
        DependencyType: "HARD"
...
```

Existe un conflicto de versiones entre dos requisitos de ComponentA:

- En esta implementación, ComponentA requiere la versión 2.5.0
- ComponentC requiere versiones de ComponentA anteriores a 2.0.0

Estos dos requisitos se contradicen entre sí, por lo que es imposible que el núcleo encuentre una versión del ComponentA que satisfaga ambos requisitos. Por lo tanto, la resolución de dependencias falla.



Ejemplo de registro de errores:

```
2025-04-11T06:07:18.291Z [ERROR] (pool-3-thread-25)
 com.aws.greengrass.componentmanager.ComponentManager: Failed to negotiate version
 with cloud and no local version to fall back to. {componentName=ComponentA,
 versionRequirement={ComponentC=<2.0.0, thinggroup/ThingGroupA==2.5.0}}
2025-04-11T06:07:18.292Z [ERROR] (pool-3-thread-24)
 com.aws.greengrass.deployment.DeploymentService: Error occurred while
 processing deployment. {deploymentId=2ffac4df-1ac9-405c-8c11-28494a1b4382,
 serviceName=DeploymentService, currentState=RUNNING}
java.util.concurrent.ExecutionException:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentA version constraints:
 ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at java.base/java.util.concurrent.FutureTask.report(FutureTask.java:122)
    at java.base/java.util.concurrent.FutureTask.get(FutureTask.java:191)
```

```
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:127)
    at
 com.aws.greengrass.deployment.DefaultDeploymentTask.call(DefaultDeploymentTask.java:50)
    at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
    at java.base/
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    at java.base/java.util.concurrent.ThreadPoolExecutor
$Worker.run(ThreadPoolExecutor.java:628)
    at java.base/java.lang.Thread.run(Thread.java:829)
Caused by:
 com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException:
 No local or cloud component version satisfies the requirements Check whether the
 version constraints conflict and that the component exists in your AWS account with
 a version that matches the version constraints. If the version constraints conflict,
 revise deployments to resolve the conflict. Component ComponentA version constraints:
 ComponentC requires <2.0.0, thinggroup/ThingGroupA requires =2.5.0.
    at
 com.aws.greengrass.componentmanager.ComponentManager.negotiateVersionWithCloud(ComponentManage
    at
 com.aws.greengrass.componentmanager.ComponentManager.resolveComponentVersion(ComponentManager.
    at com.aws.greengrass.componentmanager.DependencyResolver.lambda
$resolveDependencies$0(DependencyResolver.java:134)
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveComponentDependencies(Dependency
    at
 com.aws.greengrass.componentmanager.DependencyResolver.resolveDependencies(DependencyResolver.
    at com.aws.greengrass.deployment.DefaultDeploymentTask.lambda$call
$2(DefaultDeploymentTask.java:125)
    ... 4 more
```

Los registros indican que el núcleo no encuentra una versión de ComponentA que cumpla los siguientes requisitos.

- Los requisitos para que ComponentA sea exactamente la versión 2.5.0 (de ThingGroup A).
- El requisito de funcionar con versiones de ComponentC inferiores a 2.0.0.

Cómo resolverlo

- Si desea mantener el componente en cada grupo de objetos, seleccione la misma versión de ese componente en cada grupo de objetos.

- Seleccione una versión del componente que cumpla con los requisitos de implementación.
- Si desea utilizar una versión del componente que no cumpla los requisitos de ambos grupos de objetos, seleccione la versión del componente que cumpla con el requisito de versión del grupo de objetos y utilícelo solo en ese grupo de objetos.

> **Tip**
>
> Si ve este error en algún componente AWS proporcionado, puede resolverlo actualizando los componentes en conflicto a la última versión.

Eliminación de un dispositivo de un grupo de objetos

Cuando elimina un dispositivo principal de un grupo de objetos, el comportamiento de implementación del componente depende de la versión del núcleo de Greengrass que ejecute el dispositivo principal.

2.5.1 and later

Al eliminar un dispositivo principal de un grupo de cosas, el comportamiento depende de si la AWS IoT política concede el `greengrass:ListThingGroupsForCoreDevice` permiso. Para obtener más información sobre este permiso y AWS IoT las políticas para los dispositivos principales, consulteAutenticación y autorización de dispositivos para AWS IoT Greengrass.

- Si la AWS IoT política concede este permiso

  Al eliminar un dispositivo principal de un grupo de cosas, se AWS IoT Greengrass eliminan los componentes del grupo de cosas la próxima vez que se realice una implementación en el dispositivo. Si un componente del dispositivo se incluye en la siguiente implementación, ese componente no se elimina del dispositivo.
- Si la AWS IoT política no concede este permiso

  Cuando se elimina un dispositivo principal de un grupo de cosas, AWS IoT Greengrass no se eliminan los componentes de ese grupo de cosas del dispositivo.

  Para eliminar un componente de un dispositivo, utilice el comando crear implementación de la CLI de Greengrass. Especifique el componente que desea eliminar con el argumento `--remove` y especifique el grupo de objetos con el argumento `--groupId`.

2.5.0

Al eliminar un dispositivo principal de un grupo de cosas, se AWS IoT Greengrass eliminan los componentes del grupo de cosas la próxima vez que se realice una implementación en el dispositivo. Si un componente del dispositivo se incluye en la siguiente implementación, ese componente no se elimina del dispositivo.

Este comportamiento requiere que la AWS IoT política del dispositivo principal conceda el `greengrass:ListThingGroupsForCoreDevice` permiso. Si un dispositivo principal no tiene este permiso, no podrá aplicar las implementaciones. Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

2.0.x - 2.4.x

Cuando se elimina un dispositivo principal de un grupo de cosas, AWS IoT Greengrass no se eliminan los componentes de ese grupo de cosas del dispositivo.

Para eliminar un componente de un dispositivo, utilice el comando crear implementación de la CLI de Greengrass. Especifique el componente que desea eliminar con el argumento `--remove` y especifique el grupo de objetos con el argumento `--groupId`.

Implementaciones

Las implementaciones son continuas. Al crear una implementación, AWS IoT Greengrass despliega la implementación en los dispositivos de destino que están en línea. Si un dispositivo de destino no está en línea, recibirá la implementación la próxima vez que se conecte AWS IoT Greengrass. Al añadir un dispositivo principal a un grupo de cosas de destino, AWS IoT Greengrass envía al dispositivo la última implementación de ese grupo de cosas.

Antes de que un dispositivo principal implemente un componente, notifica de forma predeterminada a cada componente del dispositivo. Los componentes de Greengrass pueden responder a la notificación para aplazar la implementación. Es posible que desee aplazar la implementación si el nivel de batería del dispositivo es bajo o si está ejecutando un proceso que no se puede interrumpir. Para obtener más información, consulte Tutorial: Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes. Cuando crea una implementación, puede configurarla para que se implemente sin notificar a los componentes.

Cada objeto o grupo de objetos de destino puede tener una implementación a la vez. Esto significa que, al crear una implementación para un objetivo, ya AWS IoT Greengrass no se despliega la revisión anterior de la implementación de ese objetivo.

Opciones de implementación

Las implementaciones ofrecen varias opciones que le permiten controlar qué dispositivos reciben una actualización y cómo se implementa la actualización. Cuando cree una implementación, puede configurar las siguientes opciones:

- AWS IoT Greengrass componentes

  Defina los componentes que se van a instalar y ejecutar en los dispositivos de destino. AWS IoT Greengrass los componentes son módulos de software que se implementan y ejecutan en los dispositivos principales de Greengrass. Los dispositivos reciben componentes solo si el componente es compatible con la plataforma del dispositivo. Esto le permite realizar la implementación en grupos de dispositivos, incluso si los dispositivos de destino se ejecutan en varias plataformas. Si un componente no es compatible con la plataforma del dispositivo, el componente no se implementa en el dispositivo.

  Puede implementar componentes personalizados y componentes AWS proporcionados en sus dispositivos. Al implementar un componente, AWS IoT Greengrass identifica las dependencias de los componentes y también las despliega. Para obtener más información, consulte Desarrollo de componentes de AWS IoT Greengrass y Componentes proporcionados por AWS.

  Usted define la versión y la actualización de configuración que se va a implementar para cada componente. La actualización de configuración especifica cómo modificar la configuración existente del componente en el dispositivo principal o la configuración predeterminada del componente si el componente no existe en el dispositivo principal. Puede especificar qué valores de configuración desea restablecer a los valores predeterminados y los nuevos valores de configuración que se van a fusionar en el dispositivo principal. Cuando un dispositivo principal recibe implementaciones para distintos destinos y cada una especifica versiones de componentes compatibles, el dispositivo principal aplica las actualizaciones de configuración en orden según la fecha y hora en que se creó la implementación. Para obtener más información, consulte Actualización de las configuraciones de los componentes.

  > **Important**
  >
  > Al implementar un componente, AWS IoT Greengrass instala las últimas versiones compatibles de todas las dependencias de ese componente. Por este motivo, es posible que las nuevas versiones con parches de los componentes públicos AWS proporcionados se implementen automáticamente en sus dispositivos principales si agrega nuevos

dispositivos a un grupo de cosas o si actualiza la implementación destinada a esos dispositivos. Algunas actualizaciones automáticas, como las actualizaciones de núcleo, pueden provocar que los dispositivos se reinicien de forma inesperada.
Para evitar actualizaciones no deseadas de un componente que se ejecuta en su dispositivo, recomendamos que incluya directamente la versión que prefiera de ese componente cuando cree una implementación. Para obtener más información sobre el comportamiento de actualización AWS IoT Greengrass del software principal, consulteActualización del software AWS IoT Greengrass Core (OTA).

- Políticas de implementación

  Defina cuándo es seguro implementar una configuración y qué hacer si se produce un error en la implementación. Puede especificar si desea o no esperar a que los componentes informen que se pueden actualizar. También puede especificar si se debe restablecer o no los dispositivos a su configuración anterior si aplican una implementación que no funciona.

- Configuración de la detención

  Defina cómo y cuándo detener una implementación. La implementación se detiene y falla si se cumplen los criterios que usted defina. Por ejemplo, puede configurar una implementación para que se detenga si un porcentaje de dispositivos falla cuando la aplica después de recibir un número mínimo de dispositivos.

- Configuración de despliegue

  Defina la velocidad a la que una implementación se lleva a cabo en dispositivos de destino. Puede configurar un aumento de velocidad exponencial con límites de velocidad mínimos y máximos.

- Configuración del tiempo de espera

  Defina la cantidad máxima de tiempo que tiene cada dispositivo para aplicar una implementación. Si un dispositivo supera la duración que usted especifique, no podrá aplicar la implementación.

Important

Los componentes personalizados pueden definir los artefactos en los bucket de S3. Cuando el software AWS IoT Greengrass principal implementa un componente, descarga los artefactos del componente desde el Nube de AWS. Los roles de los dispositivos principales no permiten el acceso a los buckets de S3 de forma predeterminada. Para implementar componentes personalizados que definan los artefactos en un bucket de S3, el rol del

dispositivo principal debe conceder permisos para descargar los artefactos de ese bucket. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.

Temas

- Crear implementaciones
- Creación de subimplementaciones
- Revisión de las implementaciones
- Cancelación de implementaciones
- Comprobación del estado de la implementación

Crear implementaciones

Puede crear una implementación que se dirija a un objeto o un grupo de objetos.

Al crear una implementación, se configuran los componentes de software que se van a implementar y la forma en que el trabajo de implementación se extiende a los dispositivos de destino. Puede definir la implementación en el archivo JSON que proporciona a AWS CLI.

El destino de implementación determina los dispositivos en los que desea ejecutar sus componentes. Para realizar la implementación en un dispositivo principal, especifique un objeto. Para realizar la implementación en varios dispositivos principales, especifique un grupo de objetos que incluya esos dispositivos. Para obtener más información acerca de cómo configurar los grupos de objetos, consulte Grupos de objetos estáticos y Grupos de objetos dinámicos en la Guía para desarrolladores de AWS IoT.

Siga los pasos en esta sección para crear una implementación en un destino. Para obtener más información sobre cómo actualizar los componentes de software en un destino que tiene una implementación, consulte Revisión de las implementaciones.

⚠ Warning

La operación CreateDeployment puede desinstalar componentes de los dispositivos principales. Si un componente está presente en la implementación anterior y no en la nueva, el dispositivo principal desinstala ese componente. Para evitar la desinstalación de los

componentes, use primero la operación ListDeployments para comprobar si el destino de la implementación ya tiene una implementación existente. A continuación, use la operación GetDeployment para empezar desde esa implementación existente al crear una nueva implementación.

Creación de una implementación (AWS CLI)

1. Cree un archivo llamado `deployment.json` y, a continuación, copie el siguiente objeto JSON en el archivo. Reemplace *`targetArn`* por el ARN del objeto o grupo de objetos AWS IoT al que se va a destinar la implementación. Los ARN de objetos y grupos de objetos tienen el siguiente formato:

   - Cosa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
   - Grupo de cosas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

   ```
   {
     "targetArn": "targetArn"
   }
   ```

2. Compruebe si el objetivo de la implementación tiene una implementación existente que desee revisar. Haga lo siguiente:

   a. Ejecute el siguiente comando para enumerar las implementaciones del destino de implementación. Reemplace *`targetArn`* por el ARN del objeto de AWS IoT o el grupo de objetos de destino.

      ```
      aws greengrassv2 list-deployments --target-arn targetArn
      ```

      La respuesta contiene una lista con la implementación más reciente del objetivo. Si la respuesta está vacía, significa que el destino no tiene una implementación existente y puede ir directamente a Step 3. De lo contrario, copie el `deploymentId` de la respuesta para usarlo en el paso siguiente.

      > **Note**
      >
      > También puede revisar una implementación que no sea la revisión más reciente del destino. Especifique el argumento `--history-filter ALL` para enumerar todas

las implementaciones del destino. A continuación, copie el ID de la implementación que desee revisar.

b. Ejecute el siguiente comando para obtener los detalles de la implementación. Estos detalles incluyen los metadatos, los componentes y la configuración del trabajo. Reemplace *deploymentId* por el ID del paso anterior.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

La respuesta contiene los detalles de la implementación.

c. Copie cualquiera de los siguientes pares clave-valor de la respuesta del comando anterior a `deployment.json`. Puede cambiar estos valores para la nueva implementación.

- `deploymentName`: el nombre de la implementación.
- `components`: los componentes de la implementación. Para desinstalar un componente, elimínelo de este objeto.
- `deploymentPolicies`: las políticas de la implementación.
- `iotJobConfiguration`: la configuración del trabajo de la implementación.
- `tags`: las etiquetas de la implementación.

3. (Opcional) Defina un nombre para la implementación. Reemplace *deploymentName* con el nombre de la implementación.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
}
```

4. Agregue cada componente para implementar los dispositivos de destino. Para ello, agregue pares clave-valor al objeto `components`, donde la clave es el nombre del componente y el valor es un objeto que contiene los detalles de ese componente. Especifique los siguientes detalles para cada componente que agregue:

- `version`: la versión del componente que se va a implementar.
- `configurationUpdate`: la actualización de configuración que se va a implementar. La actualización es una operación de parche que modifica la configuración existente del componente en cada dispositivo de destino o la configuración predeterminada del componente

si no existe en el dispositivo de destino. Puede especificar las siguientes actualizaciones de configuración:

- Restablecer actualizaciones (`reset`): (opcional) una lista de punteros JSON que definen los valores de configuración para restablecer sus valores predeterminados en el dispositivo de destino. El software AWS IoT Greengrass Core aplica las actualizaciones de restablecimiento antes de aplicar las actualizaciones de combinación. Para obtener más información, consulte Actualizaciones de restablecimiento.
- Combinar actualizaciones (`merge`): (opcional) un documento JSON que define los valores de configuración que se van a fusionar en el dispositivo de destino. Debe serializar el documento JSON como una cadena. Para obtener más información, consulte Actualizaciones de combinación.

- `runWith`: (opcional) las opciones de proceso del sistema que usa el software AWS IoT Greengrass Core para ejecutar los procesos de este componente en el dispositivo principal. Si omite un parámetro en el objeto `runWith`, el software AWS IoT Greengrass Core usa los valores predeterminados que configure en el componente núcleo de Greengrass.

  Puede especificar cualquiera de las siguientes opciones:

  - `posixUser`: el usuario del sistema POSIX y, opcionalmente, el grupo que se usan para ejecutar este componente en dispositivos principales de Linux. El usuario y el grupo, si se especifica, deben existir en cada dispositivo principal de Linux. Especifique el usuario y el grupo separados por dos puntos (`:`) con el siguiente formato: `user:group`. El grupo es opcional. Si no especifica un grupo, el software principal de AWS IoT Greengrass utiliza el usuario principal del grupo. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.
  - `windowsUser`: el usuario de Windows que se usa para ejecutar este componente en los dispositivos principales de Windows. El usuario debe existir en cada dispositivo principal de Windows y su nombre y contraseña deben estar en la instancia del Administrador de credenciales de la cuenta LocalSystem. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

    Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.
  - `systemResourceLimits`: los límites de recursos del sistema que se aplican a los procesos de este componente. Puede aplicar límites de recursos del sistema a los

componentes de Lambda genéricos y no contenerizados. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes.

Puede especificar cualquiera de las siguientes opciones:

- `cpus`: la cantidad máxima de tiempo de CPU que los procesos de este componente pueden usar en el dispositivo principal. El tiempo total de CPU de un dispositivo principal equivale a la cantidad de núcleos de CPU del dispositivo. Por ejemplo, en un dispositivo principal con 4 núcleos de CPU, puede establecer este valor en 2 para limitar los procesos del componente al 50 % de uso de cada núcleo de CPU. En un dispositivo con 1 núcleo de CPU, puede establecer este valor en `0.25` para limitar los procesos del componente al 25 % de uso de la CPU. Si se establece este valor en un número mayor que la cantidad de núcleos de la CPU, el software AWS IoT Greengrass Core no limita el uso de la CPU del componente.
- `memory`: la cantidad máxima de RAM, expresada en kilobytes, que los procesos de un componente pueden usar en el dispositivo principal.

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Example Ejemplo de actualización de la configuración básica

El siguiente objeto `components` de muestra especifica la implementación de un componente, `com.example.PythonRuntime`, que espera un parámetro de configuración denominado `pythonVersion`.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
```

```
}
```

Example Ejemplo de actualización de configuración con actualizaciones de restablecimiento y combinación

Considere un ejemplo de componente de tablero industrial, `com.example.IndustrialDashboard`, que tiene la siguiente configuración predeterminada.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

La siguiente actualización de configuración especifica las siguientes instrucciones:

1. Restablezca la configuración HTTPS a su valor predeterminado (`true`).
2. Restablezca la lista de etiquetas industriales a una lista vacía.
3. Combine una lista de etiquetas industriales que identifiquen los flujos de datos de temperatura y presión de dos calderas.

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
```

```
  }
}
```

El siguiente objeto `components` de ejemplo especifica la implementación de este componente del tablero industrial y la actualización de la configuración.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

5. (Opcional) Defina las políticas de implementación de la implementación. Puede configurar cuándo los dispositivos principales pueden aplicar una implementación de forma segura o qué hacer si un dispositivo principal no puede aplicar la implementación. Para ello, agregue un objeto `deploymentPolicies` a `deployment.json` y, a continuación, realice una de las siguientes acciones:

   1. (Opcional) Especifique la política de actualización de los componentes (`componentUpdatePolicy`). Esta política define si la implementación permite o no a los componentes aplazar una actualización hasta que estén listos para actualizarse. Por ejemplo, es posible que los componentes deban agotar recursos o completar acciones críticas antes de poder reiniciarse para aplicar una actualización. Esta política también define el tiempo del que disponen los componentes para responder a una notificación de actualización.

      Esta política es un objeto con los siguientes parámetros:

      - `action`: (opcional) si se debe notificar o no a los componentes y esperar que informen cuando estén listos para actualizar. Puede elegir entre las siguientes opciones:

- `NOTIFY_COMPONENTS`: la implementación notifica a cada componente antes de que se detenga y actualice ese componente. Los componentes pueden usar la operación IPC SubscribeToComponentUpdates para recibir estas notificaciones.
- `SKIP_NOTIFY_COMPONENTS`: la implementación no notifica a los componentes ni espera a que se confirme que se pueden actualizar con seguridad.

El valor predeterminado es `NOTIFY_COMPONENTS`.

- `timeoutInSeconds`: la cantidad de tiempo en segundos que tiene cada componente para responder a una notificación de actualización con la operación IPC DeferComponentUpdate. Si el componente no responde en este periodo, la implementación continúa en el dispositivo principal.

  El valor predeterminado es de 60 segundos.

2. (Opcional) Especifique la política de validación de la configuración (`configurationValidationPolicy`). Esta política define la cantidad de tiempo que tiene cada componente para validar una actualización de configuración de una implementación. Los componentes pueden usar la operación IPC SubscribeToValidateConfigurationUpdates para suscribirse a las notificaciones de sus propias actualizaciones de configuración. A continuación, los componentes pueden usar la operación IPC SendConfigurationValidityReport para indicar al software AWS IoT Greengrass Core si la actualización de configuración es válida. Si la actualización de la configuración no es válida, la implementación devuelve un error.

   Esta política es un objeto con el siguiente parámetro:

   - `timeoutInSeconds`: (opcional) la cantidad de tiempo en segundos que cada componente posee para validar la actualización de configuración. Si el componente no responde en este periodo, la implementación continúa en el dispositivo principal.

     El valor predeterminado es 30 segundos.

3. (Opcional) Especifique la política de gestión de errores (`failureHandlingPolicy`). Esta política es una cadena que define si se deben revertir o no los dispositivos en caso de que se produzca un error en la implementación. Puede elegir entre las siguientes opciones:

   - `ROLLBACK`: si la implementación falla en un dispositivo principal, el software AWS IoT Greengrass Core restaura ese dispositivo principal a su configuración anterior.

- `DO_NOTHING`: si la implementación falla en un dispositivo principal, el software AWS IoT Greengrass Core conserva la nueva configuración. Esto puede provocar que los componentes se estropeen si la nueva configuración no es válida.

El valor predeterminado es `ROLLBACK`.

Su implementación en `deployment.json` puede parecerse al siguiente ejemplo:

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}
```

6. (Opcional) Defina cómo se detiene, se implementa o se agota el tiempo de espera de la implementación. AWS IoT Greengrass usa los trabajos AWS IoT Core para enviar las implementaciones a los dispositivos principales, por lo que estas opciones son idénticas a las opciones de configuración de los trabajos AWS IoT Core. Para obtener más información,

consulte la configuración de implementación y cancelación de trabajos en la Guía para desarrolladores de AWS IoT.

Para definir las opciones de trabajo, agregue un objeto `iotJobConfiguration` a `deployment.json`. A continuación, defina las opciones que desee configurar.

Su implementación en `deployment.json` puede parecerse al siguiente ejemplo:

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  },
  "iotJobConfiguration": {
    "abortConfig": {
      "criteriaList": [
        {
          "action": "CANCEL",
          "failureType": "ALL",
          "minNumberOfExecutedThings": 100,
          "thresholdPercentage": 5
        }
```

```
      ]
    },
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 5,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 10,
          "numberOfSucceededThings": 5
        }
      },
      "maximumPerMinute": 50
    },
    "timeoutConfig":  {
      "inProgressTimeoutInMinutes": 5
    }
  }
}
```

7. (Opcional) Agregue etiquetas (`tags`) para la implementación. Para obtener más información, consulte Etiquete sus AWS IoT Greengrass Version 2 recursos.
8. Ejecute el siguiente comando para crear la implementación de `deployment.json`.

   ```
   aws greengrassv2 create-deployment --cli-input-json file://deployment.json
   ```

   La respuesta incluye un `deploymentId` que identifica esta implementación. Puede usar el ID de la implementación para comprobar el estado de la implementación. Para obtener más información, consulte Comprobación del estado de la implementación.

Actualización de las configuraciones de los componentes

Las configuraciones de los componentes son objetos JSON que definen los parámetros de cada componente. La receta de cada componente define su configuración predeterminada, que puede modificar al implementar los componentes en los dispositivos principales.

Al crear una implementación, puede especificar la actualización de configuración que se aplicará a cada componente. Las actualizaciones de configuración son operaciones de parche, lo que significa que la actualización modifica la configuración del componente que existe en el dispositivo principal. Si el dispositivo principal no tiene el componente, la actualización de configuración modifica y aplica la configuración predeterminada para esa implementación.

La actualización de configuración define las actualizaciones de restablecimiento y las actualizaciones de combinación. Las actualizaciones de restablecimiento definen qué valores de configuración se deben restablecer a sus valores predeterminados o eliminar. Las actualizaciones de combinación definen los nuevos valores de configuración que se deben establecer para el componente. Al implementar una actualización de configuración, el software AWS IoT Greengrass principal ejecuta la actualización de restablecimiento antes de la actualización de fusión.

Los componentes pueden validar las actualizaciones de configuración que se implementen. El componente se suscribe para recibir una notificación cuando una implementación cambia su configuración y puede rechazar una configuración que no sea compatible. Para obtener más información, consulte Interacción con la configuración de componentes.

Temas

- Actualizaciones de restablecimiento
- Actualizaciones de combinación
- Ejemplos

Actualizaciones de restablecimiento

Las actualizaciones de restablecimiento definen qué valores de configuración se deben restablecer a sus valores predeterminados en el dispositivo principal. Si un valor de configuración no tiene un valor predeterminado, la actualización de restablecimiento elimina ese valor de la configuración del componente. Esto puede ayudarlo a reparar un componente que se interrumpe como resultado de una configuración no válida.

Utilice una lista de punteros JSON para definir qué valores de configuración desea restablecer. Los punteros JSON comienzan con una barra diagonal `/`. Para identificar un valor en una configuración de componentes anidados, use barras diagonales (`/`) para separar las claves de cada nivel de la configuración. Para obtener más información, consulte la especificación de puntero JSON.

> **Note**
>
> Solo puede restablecer los valores predeterminados de una lista completa. No puede usar las actualizaciones de restablecimiento para restablecer un elemento individual de una lista.

Para restablecer la configuración completa de un componente a sus valores predeterminados, especifique una única cadena vacía como actualización de restablecimiento.

```
"reset": [""]
```

Actualizaciones de combinación

Las actualizaciones de combinación definen los valores de configuración que se van a insertar en la configuración de los componentes principales. La actualización de fusión es un objeto JSON que el software AWS IoT Greengrass principal fusiona después de restablecer los valores de las rutas que se especificaron en la actualización de restablecimiento. Al usar AWS CLI o AWS SDKs, debe serializar este objeto JSON como una cadena.

Puede combinar un par clave-valor que no exista en la configuración predeterminada del componente. También puede combinar un par clave-valor que tenga un tipo diferente al del valor con la misma clave. El nuevo valor reemplaza al valor anterior. Esto significa que puede cambiar la estructura del objeto de configuración.

Puede combinar valores nulos y cadenas, listas y objetos vacíos.

> **Note**
>
> No puede utilizar las actualizaciones de combinación para insertar o agregar un elemento a una lista. Puede reemplazar una lista completa o definir un objeto en el que cada elemento tenga una clave única.
> AWS IoT Greengrass usa JSON para los valores de configuración. JSON especifica un tipo de número, pero no diferencia entre números enteros y flotantes. Como resultado, los valores de configuración pueden convertirse en valores flotantes en AWS IoT Greengrass. Para garantizar que su componente use el tipo de datos correcto, le recomendamos que defina los valores de configuración numéricos como cadenas. A continuación, pida a su componente que los analice como enteros o flotantes. Esto garantiza que los valores de configuración sean del mismo tipo en la configuración y en el dispositivo principal.

Uso de variables de receta en las actualizaciones de combinación

Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass.

Si establece la opción de interpolateComponentConfigurationconfiguración del núcleo de Greengrass en`true`, puede utilizar variables de receta distintas de la variable de

component_dependency_name:configuration:*json_pointer* receta en las actualizaciones de fusión. Por ejemplo, puede usar la variable de {iot:thingName} receta en una actualización de fusión para incluir el nombre del dispositivo principal en el valor de AWS IoT la configuración de un componente, como una política de autorización de comunicación entre procesos (IPC).

Ejemplos

En el siguiente ejemplo, se muestran las actualizaciones de configuración de un componente del panel que tiene la siguiente configuración predeterminada. Este componente de ejemplo muestra información sobre el equipo industrial.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

Receta de componentes para paneles industriales

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
        "useHttps": true,
        "port": {
          "http": 80,
```

```
          "https": 443
        },
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "Run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "Run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
      port:
        http: 80
        https: 443
```

```
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      Run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      Run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Example Ejemplo 1: actualización de combinación

Cree una implementación que aplique la siguiente actualización de configuración, que especifica una actualización de combinación, pero no una actualización de restablecimiento. Esta actualización de configuración indica al componente que muestre el panel en el puerto HTTP 8080 con los datos de dos calderas.

Console

Configuración de combinación

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

El siguiente comando crea una implementación a un dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

El archivo `dashboard-deployment.json` contiene el siguiente documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port
\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

El siguiente comando de la CLI de Greengrass crea una implementación local en un dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

El archivo `dashboard-configuration.json` contiene el siguiente documento JSON.

```
{
  "com.example.IndustrialDashboard": {
    "MERGE": {
      "name": "Factory 2A",
      "network": {
        "useHttps": false,
```

```
        "port": {
          "http": 8080
        }
      },
      "tags": [
        "/boiler/1/temperature",
        "/boiler/1/pressure",
        "/boiler/2/temperature",
        "/boiler/2/pressure"
      ]
    }
  }
}
```

Tras esta actualización, el componente del panel tiene la siguiente configuración.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

Example Ejemplo 2: actualizaciones de restablecimiento y combinación

A continuación, cree una implementación que aplique la siguiente actualización de configuración, que especifica una actualización de combinación y una actualización de restablecimiento. Estas actualizaciones especifican mostrar el panel en el puerto HTTPS predeterminado con datos de diferentes calderas. Estas actualizaciones modifican la configuración resultante de las actualizaciones de configuración del ejemplo anterior.

Console

Restablecer las rutas

```
[
  "/network/useHttps",
  "/tags"
]
```

Configuración de combinación

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

El siguiente comando crea una implementación a un dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

El archivo `dashboard-deployment2.json` contiene el siguiente documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
```

```
        "merge": "{\"tags\":[\"/boiler/3/temperature\",\"/boiler/3/pressure\",\"/
boiler/4/temperature\",\"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

El siguiente comando de la CLI de Greengrass crea una implementación local en un dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

El archivo `dashboard-configuration2.json` contiene el siguiente documento JSON.

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
  }
}
```

Tras esta actualización, el componente del panel tiene la siguiente configuración.

```
{
  "name": "Factory 2A",
```

```
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure",
  ]
}
```

Creación de subimplementaciones

> **Note**
>
> La característica de subimplementación está disponible en la versión 2.9.0 y posteriores del núcleo de Greengrass. No es posible implementar una configuración en una subimplementación con versiones de componentes anteriores del núcleo de Greengrass.

Una subimplementación es una implementación que se dirige a un subconjunto más pequeño de dispositivos dentro de una implementación principal. Puede usar las subimplementaciones para implementar una configuración en un subconjunto más pequeño de dispositivos. También puede crear subimplementaciones para volver a intentar una implementación principal que no funciona cuando fallan uno o más dispositivos de esa implementación principal. Con esta característica, puede seleccionar los dispositivos que fallaron en esa implementación principal y crear una subimplementación para probar las configuraciones hasta que la subimplementación se realice correctamente. Una vez que la subimplementación se haya realizado correctamente, puede volver a implementar esa configuración en la implementación principal.

Siga los pasos de esta sección para crear una subimplementación y comprobar su estado. Para obtener información sobre cómo crear implementaciones, consulte Crear implementaciones.

Creación de una subimplementación (AWS CLI)

1. Ejecute el siguiente comando para recuperar las últimas implementaciones de un grupo de objetos. Reemplace el ARN del comando por el ARN del grupo de objetos que se va a consultar. Configure `--history-filter` en **`LATEST_ONLY`** para ver la última implementación de ese grupo de objetos.

   ```
   aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
   ```

2. Copie el `deploymentId` de la respuesta al comando list-deployments para usarlo en el siguiente paso.
3. Ejecute el siguiente comando para recuperar el estado de la implementación. Reemplace *`deploymentId`* por el ID de la implementación que va a consultar.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

4. Copie el `iotJobId` de la respuesta al comando get-deployment para usarlo en el siguiente paso.
5. Ejecute el siguiente comando para recuperar la lista de ejecuciones de un trabajo especificado. Reemplace *`jobID`* por el `iotJobId` del paso anterior. Reemplace *`status`* por el estado que desee filtrar. Puede filtrar los resultados con los siguientes estados:

   - `QUEUED`
   - `IN_PROGRESS`
   - `SUCCEEDED`
   - `FAILED`
   - `TIMED_OUT`
   - `REJECTED`
   - `REMOVED`
   - `CANCELED`

   ```
   aws iot list-job-executions-for-job --job-id jobID --status status
   ```

6. Cree un grupo de objetos AWS IoT nuevo o use un grupo de objetos existente para la subimplementación. A continuación, agregue un objeto de AWS IoT a este grupo de objetos. Los

grupos de objetos se usan para administrar las flotas de dispositivos principales de Greengrass. Al implementar componentes de software en sus dispositivos, puede dirigirse a dispositivos individuales o a grupos de dispositivos. Puede agregar un dispositivo a un grupo de objetos con una implementación activa de Greengrass. Una vez agregado, puede implementar los componentes de software de ese grupo de objetos en ese dispositivo.

Para crear un nuevo grupo de objetos y agregarle sus dispositivos, haga lo siguiente:

a. Cree un grupo de objetos de AWS IoT. Reemplace *MyGreengrassCoreGroup* por el nombre del nuevo grupo de objetos. No puede usar dos puntos (:) en el nombre de un grupo de objetos.

> **Note**
>
> Si un grupo de objetos de una subimplementación se usa con un `parentTargetArn`, no se puede reutilizar con una flota principal diferente. Si un grupo de objetos ya se ha usado para crear una subimplementación para otra flota, la API devolverá un error.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

Si la solicitud se realiza correctamente, la respuesta tiene un aspecto similar al del siguiente ejemplo:

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. Agregue un núcleo de Greengrass aprovisionado a su grupo de objetos. Ejecute el siguiente comando con estos parámetros:

- Reemplace *MyGreengrassCore* por el nombre del núcleo de Greengrass aprovisionado.
- Reemplace *MyGreengrassCoreGroup* por el nombre su grupo de objetos.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

7. Cree un archivo llamado `deployment.json` y, a continuación, copie el siguiente objeto JSON en el archivo. Reemplace *`targetArn`* por el ARN del grupo de objetos AWS IoT al que apuntar la subimplementación. Un objetivo de subimplementación solo puede ser un grupo de objetos. Los ARN del grupo de objetos tienen el siguiente formato:

   - Grupo de objetos: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

   ```
   {
     "targetArn": "targetArn"
   }
   ```

8. Ejecute el siguiente comando nuevamente para obtener los detalles originales de la implementación. Estos detalles incluyen los metadatos, los componentes y la configuración del trabajo. Reemplace *`deploymentId`* por el ID de Step 1. Puede usar esta configuración de implementación para configurar su subimplementación y realizar los cambios necesarios.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   La respuesta contiene los detalles de la implementación. Copie cualquiera de los siguientes pares clave-valor de la respuesta del comando get-deployment en `deployment.json`. Puede cambiar estos valores para la subimplementación. Para obtener más información acerca de los detalles de este comando, consulte GetDeployment.

   - `components`: los componentes de la implementación. Para desinstalar un componente, elimínelo de este objeto.
   - `deploymentName`: el nombre de la implementación.
   - `deploymentPolicies`: las políticas de la implementación.
   - `iotJobConfiguration`: la configuración del trabajo de la implementación.
   - `parentTargetArn`: el objetivo de la implementación principal.
   - `tags`: las etiquetas de la implementación.

9. Ejecute el siguiente comando para crear la subimplementación desde `deployment.json`. Reemplace *`subdeploymentName`* por un nombre para la subimplementación.

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-
json file://deployment.json
```

   La respuesta incluye un `deploymentId` que identifica esta subimplementación. Puede usar el ID de la implementación para comprobar el estado de la implementación. Para obtener más información, consulte Comprobar el estado de la implementación.

10. Si la subimplementación se realiza correctamente, puede usar su configuración para revisar la implementación principal. Copie el `deployment.json` que usó en el paso anterior. Reemplace el `targetArn` del archivo JSON por el ARN de la implementación principal y ejecute el siguiente comando para crear la implementación principal con esta nueva configuración.

> **Note**
>
> Si crea una nueva revisión de implementación de la flota principal, reemplaza a todas las revisiones y subimplementaciones de la implementación principal. Para obtener más información, consulte Revisar implementaciones.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

   La respuesta incluye un `deploymentId` que identifica esta implementación. Puede usar el ID de la implementación para comprobar el estado de la implementación. Para obtener más información, consulte Comprobación del estado de la implementación.

Revisión de las implementaciones

Cada objeto o grupo de objetos de destino puede tener una implementación a la vez. Al crear una implementación para un destino que ya tiene una implementación, los componentes de software de la nueva implementación reemplazan a los de la implementación anterior. Si la nueva implementación no define un componente que definió la implementación anterior, el software AWS IoT Greengrass Core elimina ese componente de los dispositivos principales de destino. Puede revisar una implementación existente para no eliminar los componentes que se ejecutan en los dispositivos principales de una implementación anterior a un destino.

Para revisar una implementación, debe crear una implementación que comience con los mismos componentes y configuraciones que existían en una implementación anterior. Se utiliza la operación CreateDeployment, que es la misma operación que se utiliza para crear implementaciones.

Cómo revisar una implementación (AWS CLI)

1. Ejecute el siguiente comando para enumerar las implementaciones del destino de implementación. Reemplace *targetArn* por el ARN del objeto de AWS IoT o el grupo de objetos de destino.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   La respuesta contiene una lista con la implementación más reciente del objetivo. Copie el `deploymentId` de la respuesta para usarlo en el siguiente paso.

   > **Note**
   >
   > También puede revisar una implementación que no sea la revisión más reciente del destino. Especifique el argumento `--history-filter ALL` para enumerar todas las implementaciones del destino. A continuación, copie el ID de la implementación que desee revisar.

2. Ejecute el siguiente comando para obtener los detalles de la implementación. Estos detalles incluyen los metadatos, los componentes y la configuración del trabajo. Reemplace *deploymentId* por el ID del paso anterior.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   La respuesta contiene los detalles de la implementación.

3. Cree un archivo llamado `deployment.json` y copie la respuesta del comando anterior en el archivo.

4. Elimine los siguientes pares de clave-valor del objeto JSON en `deployment.json`:

   - `deploymentId`
   - `revisionId`
   - `iotJobId`
   - `iotJobArn`

- `creationTimestamp`
- `isLatestForTarget`
- `deploymentStatus`

La operación CreateDeployment espera una carga útil con la siguiente estructura.

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. En `deployment.json`, realice una de las siguientes acciones:
   - Cambie el nombre de la implementación (`deploymentName`).
   - Cambie los componentes de la implementación (`components`).
   - Cambie las políticas de la implementación (`deploymentPolicies`).
   - Cambie la configuración del trabajo de la implementación (`iotJobConfiguration`).
   - Cambie las etiquetas de la implementación (`tags`).

   Para obtener más información acerca de cómo definir los detalles de la implementación, consulte Crear implementaciones.
6. Ejecute el siguiente comando para crear la implementación de `deployment.json`.

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

   La respuesta incluye un `deploymentId` que identifica esta implementación. Puede usar el ID de la implementación para comprobar el estado de la implementación. Para obtener más información, consulte Comprobación del estado de la implementación.

Cancelación de implementaciones

Puede cancelar una implementación activa para evitar que sus componentes de software se instalen en los dispositivos principales de AWS IoT Greengrass. Si cancela una implementación dirigida a un grupo de objetos, los dispositivos principales que agregue al grupo no recibirán esa implementación

continua. Si un dispositivo principal ya ejecuta la implementación, no cambiará los componentes de ese dispositivo cuando cancele la implementación. Debe crear una nueva implementación o revisar la implementación para modificar los componentes que se ejecutan en los dispositivos principales que recibieron la implementación cancelada.

Cancelación de una implementación (AWS CLI)

1. Ejecute el siguiente comando para encontrar el ID de la última revisión de implementación de un destino. La última revisión es la única implementación que puede estar activa para un destino, ya que las implementaciones anteriores se cancelan al crear una nueva revisión. Reemplace *targetArn* por el ARN del objeto de AWS IoT o el grupo de objetos de destino.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   La respuesta contiene una lista con la implementación más reciente del objetivo. Copie el `deploymentId` de la respuesta para usarlo en el siguiente paso.

2. Ejecute el siguiente comando para cancelar la implementación. Reemplace *deploymentId* por el ID del paso anterior.

   ```
   aws greengrassv2 cancel-deployment --deployment-id deploymentId
   ```

   Si la operación es exitosa, el estado de la implementación cambia a `CANCELED`.

Comprobación del estado de la implementación

Puede comprobar el estado de una implementación que ha creado en AWS IoT Greengrass. También puede comprobar el estado de los trabajos de AWS IoT que hacen la implementación en cada dispositivo principal. Mientras una implementación está activa, el estado del trabajo de AWS IoT es `IN_PROGRESS`. Tras crear una nueva revisión de una implementación, el estado del trabajo de AWS IoT de la revisión anterior cambia a `CANCELLED`.

Temas

- Comprobación del estado de la implementación
- Comprobación del estado de la implementación del dispositivo

Comprobación del estado de la implementación

Puede comprobar el estado de una implementación que identifique por su objetivo o su ID.

Comprobación del estado de la implementación por destino (AWS CLI)

- Ejecute el siguiente comando para recuperar el estado de la implementación más reciente por destino. Reemplace *targetArn* por el nombre de recurso de Amazon (ARN) del objeto o grupo de objetos de AWS IoT que la implementación tiene como destino.

  ```
  aws greengrassv2 list-deployments --target-arn targetArn
  ```

  La respuesta contiene una lista con la implementación más reciente del objetivo. Este objeto de implementación incluye el estado de la implementación.

Comprobación del estado de la implementación por ID (AWS CLI)

- Ejecute el siguiente comando para recuperar el estado de la implementación. Reemplace *deploymentId* por el ID de la implementación que va a consultar.

  ```
  aws greengrassv2 get-deployment --deployment-id deploymentId
  ```

  La respuesta contiene el estado de la implementación.

Comprobación del estado de la implementación del dispositivo

Puede comprobar el estado de un trabajo de implementación que se aplica a un dispositivo principal individual. También puede comprobar el estado de un trabajo de implementación para la implementación de un grupo de objetos.

Comprobación de los estados del trabajo de implementación de un dispositivo principal (AWS CLI)

- Ejecute el siguiente comando para recuperar el estado de todos los trabajos de implementación para un dispositivo principal. Reemplace *coreDeviceName* por el nombre del dispositivo principal que se va a consultar.

  ```
  aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
  ```

La respuesta contiene la lista de trabajos de implementación para el dispositivo principal. Puede identificar el trabajo de una implementación por el `deploymentId` o `targetArn` del trabajo. Cada trabajo de implementación contiene el estado del trabajo en el dispositivo principal.

Comprobación de los estados de implementación de un grupo de objetos (AWS CLI)

1. Ejecute el siguiente comando para recuperar el ID de una implementación existente. Reemplace *`targetArn`* por el ARN del grupo de objetos de destino.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   La respuesta contiene una lista con la implementación más reciente del objetivo. Copie el `deploymentId` de la respuesta para usarlo en el siguiente paso.

   > **Note**
   >
   > También puede enumerar una implementación que no sea la implementación más reciente del destino. Especifique el argumento `--history-filter ALL` para enumerar todas las implementaciones del destino. A continuación, copie el ID de la implementación cuyo estado desea comprobar.

2. Ejecute el siguiente comando para obtener los detalles de la implementación. Reemplace *`deploymentID`* por el ID del paso anterior.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   La respuesta contiene información sobre la implementación. Copie el `iotJobId` de la respuesta para usarlo en el siguiente paso.

3. Ejecute el siguiente comando para describir la ejecución del trabajo de un dispositivo principal para la implementación. Reemplace *`iotJobId`* y *`coreDeviceThingName`* por el ID de trabajo del paso anterior y el dispositivo principal cuyo estado desea comprobar.

   ```
   aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
   ```

   La respuesta contiene el estado de la ejecución del trabajo de implementación del dispositivo principal y detalles sobre el estado. El `detailsMap` contiene la siguiente información:

- `detailed-deployment-status`: el estado del resultado de la implementación, que puede ser uno de los siguientes valores:
  - `SUCCESSFUL`: la implementación se realizó correctamente.
  - `FAILED_NO_STATE_CHANGE`: la implementación falló mientras el dispositivo principal se preparaba para aplicarla.
  - `FAILED_ROLLBACK_NOT_REQUESTED`: la implementación falló y esta no especificó volver a una configuración de trabajo anterior, por lo que es posible que el dispositivo principal no funcione correctamente.
  - `FAILED_ROLLBACK_COMPLETE`: la implementación falló y el dispositivo principal se restableció correctamente a una configuración de trabajo anterior.
  - `FAILED_UNABLE_TO_ROLLBACK`: la implementación falló y el dispositivo principal no pudo volver a una configuración de trabajo anterior, por lo que es posible que el dispositivo principal no funcione correctamente.

  Si la implementación falló, compruebe el valor `deployment-failure-cause` y los archivos de registro del dispositivo principal para identificar el problema. Para obtener más información sobre cómo acceder a los archivos de registro del dispositivo principal, consulte Supervisión de los registros de AWS IoT Greengrass.
- `deployment-failure-cause`: un mensaje de error que proporciona detalles adicionales sobre el motivo por el que se ha producido un error en la ejecución del trabajo.

La respuesta tiene un aspecto similar a la del siguiente ejemplo.

```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version satisfies
 the requirements. Check whether the version constraints conflict and that
 the component exists in your Cuenta de AWS with a version that matches the
 version constraints. If the version constraints conflict, revise deployments
 to resolve the conflict. Component com.example.HelloWorld version constraints:
 LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
 =1.0.1.",
        "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
```

```
      }
    },
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
    "queuedAt": "2022-02-15T14:45:53.098000-08:00",
    "startedAt": "2022-02-15T14:46:05.670000-08:00",
    "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
    "executionNumber": 1,
    "versionNumber": 3
  }
}
```

Registro y monitorización en AWS IoT Greengrass

La supervisión es un aspecto importante del mantenimiento de la fiabilidad, la disponibilidad y el rendimiento de AWS IoT Greengrass y sus soluciones de AWS. Debe recopilar datos de monitorización de todas las partes de su solución de AWS para que le resulte más sencillo depurar cualquier error que se produzca en distintas partes del código, en caso de que ocurra. Antes de empezar a monitorear AWS IoT Greengrass, debe crear un plan de monitoreo que incluya respuestas a las siguientes preguntas:

- ¿Cuáles son los objetivos de la monitorización?
- ¿Qué recursos va a monitorizar?
- ¿Con qué frecuencia va a monitorizar estos recursos?
- ¿Qué herramientas de monitorización va a utilizar?
- ¿Quién se encargará de realizar las tareas de supervisión?
- ¿Quién debería recibir una notificación cuando surjan problemas?

Temas

- Herramientas de supervisión
- Supervisión de los registros de AWS IoT Greengrass
- Registra las llamadas a la AWS IoT Greengrass V2 API con AWS CloudTrail
- Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass
- Reciba notificaciones sobre el estado de la implementación y de los componentes
- Comprobación del estado del dispositivo principal de Greengrass

Herramientas de supervisión

AWS proporciona herramientas que puede utilizar para monitorizar AWS IoT Greengrass. Puede configurar algunas de estas herramientas para que realicen la monitorización por usted. Algunas de las herramientas requieren intervención manual. Le recomendamos que automatice las tareas de supervisión en la medida de lo posible.

Puede utilizar las siguientes herramientas de monitorización automatizada para monitorizar AWS IoT Greengrass e informar de los problemas:

- Registros de Amazon CloudWatch: monitoree, almacene y obtenga acceso a los archivos de registro de AWS CloudTrail u otras fuentes. Para obtener más información, consulte Monitoreo de archivos de registro en la Guía del usuario de Amazon CloudWatch.
- Monitoreo de registros de AWS CloudTrail: comparta archivos de registro entre cuentas, monitoree los archivos de registro de CloudTrail en tiempo real enviándolos a CloudWatch Logs, escriba aplicaciones de procesamiento de registros en Java y compruebe que los archivos de registro no hayan cambiado después de que CloudTrail los entregara. Para obtener más información, consulte Uso de archivos de registro de CloudTrail en la Guía del usuario de AWS CloudTrail.
- Telemetría de salud del sistema Greengrass: suscríbase para recibir los datos de telemetría enviados desde el núcleo de Greengrass. Para obtener más información, consulte the section called “Recopilación de datos de telemetría del estado del sistema”.
- Notificaciones de estado del dispositivo Cree eventos con Amazon EventBridge para recibir actualizaciones de estado relacionadas con las implementaciones y los componentes. Para obtener más información, consulte Reciba notificaciones sobre el estado de la implementación y de los componentes.
- Servicio de estado de la flota: utilice las operaciones de la API de estado de la flota para comprobar el estado de los dispositivos principales y sus componentes de Greengrass. También puede ver la información del estado de la flota en la consola de AWS IoT Greengrass. Para obtener más información, consulte Comprobación del estado del dispositivo principal de Greengrass.

Supervisión de los registros de AWS IoT Greengrass

AWS IoT Greengrass consta del servicio de nube y el software de AWS IoT Greengrass Core. El software AWS IoT Greengrass Core puede escribir Registros de Amazon CloudWatch y en el sistema de archivos local del dispositivo principal. Los componentes de Greengrass que se ejecutan en el dispositivo principal también pueden escribir registros en los Registros de CloudWatch y en el sistema de archivos local. Puede utilizar registros para monitorizar eventos y solucionar problemas. Todas las entradas de registro de AWS IoT Greengrass incluyen una marca temporal, un nivel de registro e información sobre el evento.

De forma predeterminada, el software AWS IoT Greengrass Core escribe los registros únicamente en el sistema de archivos local. Puede ver los registros del sistema de archivos en tiempo real para poder depurar los componentes de Greengrass que desarrolle e implemente. También puede configurar un dispositivo principal para escribir registros en los Registros de CloudWatch, de modo

que pueda solucionar los problemas del dispositivo principal sin acceso al sistema de archivos local. Para obtener más información, consulte Habilitación del registro en los Registros de CloudWatch.

Temas

- Acceso a los registros del sistema de archivos
- Acceso a los Registros de CloudWatch
- Acceso a los registros de servicios del sistema
- Habilitación del registro en los Registros de CloudWatch
- Configuración de registro en AWS IoT Greengrass
- AWS CloudTrailRegistros de

Acceso a los registros del sistema de archivos

El software AWS IoT Greengrass Core almacena los registros en la carpeta */greengrass/v2/*`logs` de un dispositivo principal, donde */greengrass/v2* es la ruta a la carpeta raíz de AWS IoT Greengrass. La carpeta de registros tiene la estructura siguiente.

```
/greengrass/v2
### logs
    ### greengrass.log
    ### greengrass_2021_09_14_15_0.log
    ### ComponentName.log
    ### ComponentName_2021_09_14_15_0.log
    ### main.log
```

- `greengrass.log`: el archivo de registro del software AWS IoT Greengrass Core. Utilice este archivo de registro para ver información en tiempo real sobre los componentes y las implementaciones. Este archivo de registro incluye registros del núcleo de Greengrass, que es el núcleo del software AWS IoT Greengrass Core, y componentes del complemento, como el administrador de registros y el administrador de secretos.
- *ComponentName*`.log`: archivos de registro de componentes de Greengrass. Utilice los archivos de registro de componentes para ver información en tiempo real sobre un componente de Greengrass que se ejecuta en el dispositivo principal. Los componentes genéricos y los componentes de Lambda escriben la salida estándar (stdout) y el error estándar (stderr) en estos archivos de registro.

- `main.log`: el archivo de registro del servicio `main` que gestiona los ciclos de vida de los componentes. Este archivo de registro siempre estará vacío.

Para obtener más información acerca de las diferencias entre los componentes de complemento, genéricos y Lambda, consulte Tipos de componentes.

Las siguientes consideraciones se aplican cuando se utilizan los registros del sistema de archivos:

- Permisos de usuario raíz

  Debe tener permisos de raíz para leer registros de AWS IoT Greengrass en el sistema de archivos.

- Rotación de archivos de registro

  El software AWS IoT Greengrass Core rota los archivos de registro cada hora o cuando superan un límite de tamaño de archivo. Los archivos de registro rotados contienen una marca temporal en el nombre del archivo. Por ejemplo, un archivo de registro del software AWS IoT Greengrass Core rotado podría tener un nombre `greengrass_2021_09_14_15_0.log`. El límite de tamaño de archivo predeterminado es de 1024 KB (1 MB). Puede configurar el límite de tamaño de archivo en el componente núcleo de Greengrass.

- Eliminación del archivo de registro

  El software AWS IoT Greengrass Core borra los archivos de registro anteriores cuando el tamaño de los archivos de registro del software AWS IoT Greengrass Core o los archivos de registro de los componentes de Greengrass, incluidos los archivos de registro rotados, excede un límite de espacio en disco. El límite de espacio en disco predeterminado para el registro del software AWS IoT Greengrass Core y para cada registro de componentes es de 10 240 KB (10 MB). Puede configurar el límite de espacio en disco de registro del software AWS IoT Greengrass Core en el componente núcleo de Greengrass o en el componente administrador de registros. Puede configurar el límite de espacio en disco de registro de cada componente en el componente administrador de registros.

Cómo ver el archivo de registro del software AWS IoT Greengrass Core

- Ejecute el siguiente comando para ver el archivo de registro en tiempo real. Sustituya */greengrass/v2* por la ruta a la carpeta raíz de AWS IoT Greengrass.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

El comando `type` escribe el contenido del archivo en la terminal. Ejecute este comando varias veces para observar los cambios en el archivo.

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Cómo ver el archivo de registro de un componente

- Ejecute el siguiente comando para ver el archivo de registro en tiempo real. Sustituya */greengrass/v2* o *C:\greengrass\v2* por la ruta a la carpeta raíz de AWS IoT Greengrass y sustituya *com.example.HelloWorld* por el nombre del componente.

  Linux or Unix

  ```
  sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
  ```

  PowerShell

  ```
  gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
  ```

También puede usar el comando `logs` de la CLI de Greengrass para analizar los registros de Greengrass en un dispositivo principal. Para usar el comando `logs`, debe configurar el núcleo de Greengrass para que genere archivos de registro en formato JSON. Para obtener más información, consulte Interfaz de la línea de comandos de Greengrass y registros.

Acceso a los Registros de CloudWatch

Puede implementar el componente administrador de registros para configurar el dispositivo principal para que escriba en los Registros de CloudWatch. Para obtener más información, consulte Habilitación del registro en los Registros de CloudWatch. Luego, puede ver los registros en la página Registros de la consola de Amazon CloudWatch o mediante la API de Registros de CloudWatch.

Nombre del grupo de registro

```
/aws/greengrass/componentType/region/componentName
```

El nombre del grupo de registro utiliza las siguientes variables:

- `componentType`: el tipo del componente, que puede ser uno de los siguientes:
  - `GreengrassSystemComponent`: este grupo de registro incluye los registros de los componentes del núcleo y del complemento, que se ejecutan en la misma JVM que el núcleo de Greengrass. El componente forma parte del núcleo de Greengrass.
  - `UserComponent`: este grupo de registro incluye registros de componentes genéricos, componentes de Lambda y otras aplicaciones del dispositivo. El componente no forma parte del núcleo de Greengrass.

  Para obtener más información, consulte Tipos de componentes.
- `region`: la región de AWS que utiliza el dispositivo principal.
- `componentName`: el nombre del componente. En el caso de los registros del sistema, este valor es `System`.

Nombre del flujo de registro

```
/date/thing/thingName
```

El nombre del flujo de registro utiliza las siguientes variables:

- `date`: la fecha del registro, por ejemplo, `2020/12/15`. El componente administrador de registros usa el formato `yyyy/MM/dd`.
- `thingName`: el nombre del dispositivo principal.

> **Note**
>
> Si el nombre de un objeto contiene dos puntos (`:`), el administrador de registros los sustituye por un signo más (+).

Las siguientes consideraciones se aplican cuando utilice el administrador de registros para escribir en Registros de CloudWatch:

- Demoras de registro

> **Note**
>
> Le recomendamos que actualice a la versión 2.3.0 del administrador de registros, ya que reduce las demoras en el registro de los archivos de registro activos y rotados. Cuando actualice al administrador de registros 2.3.0, le recomendamos que también actualice al núcleo de Greengrass 2.9.1.

  La versión 2.2.8 (y anteriores) del componente administrador de registros procesa y carga los registros únicamente a partir de archivos de registro rotados. De forma predeterminada, el software AWS IoT Greengrass Core rota los archivos de registro cada hora o después de que ocupen 1024 KB. Como resultado, el componente del administrador de registros carga los registros solo después de que el software AWS IoT Greengrass Core o un componente de Greengrass hayan escrito registros con un valor superior a 1024 KB. Puede configurar un límite de tamaño de archivo de registro inferior para que los archivos de registro roten con mayor frecuencia. Esto hace que el componente del administrador de registros cargue registros en los Registros de CloudWatch con mayor frecuencia.

  La versión 2.3.0 (y posteriores) del componente administrador de registros procesa y carga todos los registros. Cuando escribe un registro nuevo, la versión 2.3.0 (y posteriores) del administrador de registros procesa y carga directamente el archivo de registro activo en lugar de esperar a que se rote. Esto significa que puede ver el nuevo registro en 5 minutos o menos.

  El componente administrador de registros carga nuevos registros periódicamente. De forma predeterminada, el componente del administrador de registros carga nuevos registros cada 5 minutos. Puede configurar un intervalo de carga inferior para que el componente administrador de registros cargue los registros en los Registros de CloudWatch con mayor frecuencia si configura

`periodicUploadIntervalSec`. Para obtener más información acerca de cómo configurar este intervalo periódico, consulte Configuración.

Los registros se pueden cargar prácticamente en tiempo real desde el mismo sistema de archivos de Greengrass. Si necesita observar los registros en tiempo real, considere usar los registros del sistema de archivos.

> **Note**
>
> Si utiliza distintos sistemas de archivos para escribir los registros, el administrador de registros vuelve al comportamiento de las versiones 2.2.8 y anteriores del componente administrador de registros. Para obtener información sobre cómo acceder a los registros del sistema de archivos, consulte Acceder a los registros del sistema de archivos.

- Desfase del reloj

  El componente administrador de registros utiliza el proceso de firma estándar de la versión 4 de Signature para crear solicitudes de API a los Registros de CloudWatch. Si la hora del sistema del dispositivo principal no está sincronizada en más de 15 minutos, los Registros de CloudWatch rechaza las solicitudes. Para obtener más información, consulte Proceso de firma Signature Version 4 en la Referencia general de AWS.

Acceso a los registros de servicios del sistema

Si configura el software AWS IoT Greengrass Core como un servicio del sistema, puede ver los registros de servicio del sistema para solucionar problemas, como el hecho de que el software no se inicie.

Cómo ver los registros de servicio del sistema (CLI)

1. Ejecute el siguiente comando para ver registros de servicio del sistema de software AWS IoT Greengrass Core.

   Linux or Unix (systemd)

   ```
   sudo journalctl -u greengrass.service
   ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.wrapper.log
```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.wrapper.log
```

2. En los dispositivos Windows, el software AWS IoT Greengrass Core crea un archivo de registro independiente para los errores del servicio del sistema. Ejecute el siguiente comando para ver los registros de errores del servicio del sistema.

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\greengrass.err.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.err.log
   ```

En los dispositivos Windows, también puede utilizar la aplicación Event Viewer para ver los registros de servicio del sistema.

Cómo ver los registros de servicio de Windows (Event Viewer)

1. Abra la aplicación Event Viewer.
2. Seleccione Registros de Windows para expandirlo.
3. Elija Aplicación para ver los registros de servicio de la aplicación.
4. Busque y abra los registros de eventos cuyo Origen sea greengrass.

Habilitación del registro en los Registros de CloudWatch

Puede implementar el componente administrador de registros para configurar el dispositivo principal para que escriba registros en los Registros de CloudWatch. Puede habilitar los Registros de CloudWatch para los registros del software AWS IoT Greengrass Core y puede habilitar los Registros de CloudWatch para componentes específicos de Greengrass.

> **Note**
>
> El rol de intercambio de token del dispositivo principal de Greengrass debe permitir que el dispositivo principal escriba en los Registros de CloudWatch, como se muestra en el siguiente ejemplo de política de IAM. Si instaló el software AWS IoT Greengrass Core con el aprovisionamiento automático de recursos, su dispositivo principal tiene estos permisos.
>
> ```
> {
>   "Version": "2012-10-17",
>   "Statement": [
>     {
>       "Action": [
>         "logs:CreateLogGroup",
>         "logs:CreateLogStream",
>         "logs:PutLogEvents",
>         "logs:DescribeLogStreams"
>       ],
>       "Effect": "Allow",
>       "Resource": "arn:aws:logs:*:*:*"
>     }
>   ]
> }
> ```

Para configurar un dispositivo principal para escribir registros del software AWS IoT Greengrass Core en los Registros de CloudWatch, cree una implementación que especifique una actualización de configuración que establezca `uploadToCloudWatch` en `true` para el componente `aws.greengrass.LogManager`. AWS IoT Greengrass Los registros del software Core incluyen los registros del núcleo de Greengrass y los componentes del complemento.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Para configurar un dispositivo principal para escribir los registros de un componente de Greengrass en los Registros de CloudWatch, cree una implementación que especifique una actualización de

configuración que agregue el componente a la lista de configuraciones de registro de componentes. Al agregar un componente a esta lista, el componente administrador de registros escribe registros en los Registros de CloudWatch. Los registros de componentes incluyen registros de componentes genéricos y componentes de Lambda.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
  }
}
```

Al implementar el componente administrador de registros, también puede configurar los límites de espacio en disco y si el dispositivo principal eliminará los archivos de registro después de escribirlos en los Registros de CloudWatch. Para obtener más información, consulte Configuración de registro en AWS IoT Greengrass.

Configuración de registro en AWS IoT Greengrass

Puede configurar las siguientes opciones para personalizar el registro de los dispositivos principales de Greengrass. Para configurar estas opciones, cree una implementación que especifique una actualización de configuración para los componentes del núcleo de Greengrass o del administrador de registros.

- Escritura de registros en los Registros de CloudWatch

  Para solucionar problemas de dispositivos principales de forma remota, puede configurarlos para que escriban registros de componentes y del software AWS IoT Greengrass Core en los Registros de CloudWatch. Para ello, implemente y configure el componente administrador de registros. Para obtener más información, consulte Habilitación del registro en los Registros de CloudWatch.

- Eliminación de los archivos de registro cargados

  Para reducir el uso del espacio en disco, puede configurar los dispositivos principales para que eliminen los archivos de registro después de escribirlos en los Registros de CloudWatch. Para obtener más información, consulte el parámetro `deleteLogFileAfterCloudUpload` del

componente administrador de registros, que puede especificar para los registros del software AWS IoT Greengrass Core y los registros de los componentes.

- Límites de espacio en disco de registro

  Para limitar el uso del espacio en disco, puede configurar el espacio máximo en disco para cada registro, incluidos sus archivos de registro rotados, en un dispositivo principal. Por ejemplo, puede configurar el espacio máximo combinado en disco para `greengrass.log` y los archivos rotados `greengrass.log`. Para obtener más información, consulte el parámetro `logging.totalLogsSizeKB` del componente núcleo de Greengrass y el parámetro `diskSpaceLimit` del componente administrador de registros, que puede especificar para los registros del software AWS IoT Greengrass Core y registros de componente.

- Límite de tamaño de los archivos de registro

  Puede configurar el tamaño máximo de archivo para cada archivo de registro. Cuando un archivo de registro supera este límite de tamaño de archivo, el software AWS IoT Greengrass Core crea un nuevo archivo de registro. El componente administrador de registros versión 2.28 (y anteriores) solo escribe archivos de registro rotados en los Registros de CloudWatch, por lo que puede especificar un límite de tamaño de archivo inferior para escribir registros en los Registros de CloudWatch con mayor frecuencia. La versión 2.3.0 (y posteriores) del componente administrador de registros procesa y carga todos los registros en lugar de esperar a que se roten. Para obtener más información, consulte el parámetro de límite de tamaño de archivo de registro del componente de núcleo de Greengrass (`logging.fileSizeKB`).

- Niveles mínimos de registro

  Puede configurar el nivel de registro mínimo que el componente núcleo de Greengrass escribe en los registros del sistema de archivos. Por ejemplo, puede especificar los registros de `DEBUG` para facilitar la solución de problemas, o puede especificar los registros de `ERROR` para reducir la cantidad de registros que crea un dispositivo principal. Para obtener más información, consulte el parámetro de límite de registro del componente núcleo de Greengrass (`logging.level`).

  También puede configurar el nivel de registro mínimo que el componente del administrador de registros escribe en los Registros de CloudWatch. Por ejemplo, puede especificar un nivel de registro superior para reducir los costos de registro. Para obtener más información, consulte el parámetro `minimumLogLevel` del componente administrador de registros, que puede especificar para los registros del software AWS IoT Greengrass Core y los registros de los componentes.

- Intervalo para comprobar los registros que se van a escribir en los Registros de CloudWatch

Para aumentar o reducir la frecuencia con la que el componente administrador de registros escribe registros en los Registros de CloudWatch, puede configurar el intervalo en el que comprueba si hay nuevos archivos de registro que escribir. Por ejemplo, puede especificar un intervalo inferior para ver los registros en los Registros de CloudWatch antes de lo que lo haría con el intervalo predeterminado de 5 minutos. Puede especificar un intervalo mayor para reducir los costos, ya que el componente administrador de registros agrupa los archivos de registro en menos solicitudes. Para obtener más información, consulte el parámetro de intervalo de carga (`periodicUploadIntervalSec`) del componente administrador de registros.

- Formato de registro

  Puede elegir si el software AWS IoT Greengrass Core escribe registros en formato de texto o JSON. Elija el formato de texto si lee los registros o el formato JSON si utiliza una aplicación para leer o analizar los registros. Para obtener más información, consulte el parámetro de formato de registro (`logging.format`) del componente núcleo de Greengrass.

- Carpeta de registros del sistema de archivos local

  Puede cambiar la carpeta de registros */greengrass/v2*`/logs` por otra carpeta del dispositivo principal. Para obtener más información, consulte el parámetro del directorio de salida (`logging.outputDirectory`) del componente núcleo de Greengrass.

AWS CloudTrailRegistros de

AWS IoT Greengrass se integra con AWS CloudTrail, un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o Servicio de AWS en AWS IoT Greengrass. Para obtener más información, consulte Registra las llamadas a la AWS IoT Greengrass V2 API con AWS CloudTrail.

Registra las llamadas a la AWS IoT Greengrass V2 API con AWS CloudTrail

AWS IoT Greengrass V2 está integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un AWS servicio en él AWS IoT Greengrass Version 2. CloudTrail captura todas las llamadas a la API AWS IoT Greengrass como eventos. Las llamadas que se capturan incluyen las llamadas desde la AWS IoT Greengrass consola y las llamadas en código a las operaciones de la AWS IoT Greengrass API.

Si crea un registro, puede habilitar la entrega continua de CloudTrail eventos a un bucket de S3, incluidos los eventos correspondientes AWS IoT Greengrass. Si no configuras una ruta, podrás ver los eventos más recientes en la CloudTrail consola, en el historial de eventos. Con la información recopilada por usted CloudTrail, puede determinar el destinatario de la solicitud AWS IoT Greengrass, la dirección IP desde la que se realizó la solicitud, quién la realizó, cuándo se realizó y detalles adicionales.

Para obtener más información al respecto CloudTrail, consulte la Guía AWS CloudTrail del usuario.

Temas

- AWS IoT Greengrass V2 información en CloudTrail
- AWS IoT Greengrass eventos de datos en CloudTrail
- AWS IoT Greengrass eventos de gestión en CloudTrail
- Descripción de las entradas de los archivos de AWS IoT Greengrass V2 registro

AWS IoT Greengrass V2 información en CloudTrail

CloudTrail está habilitada en tu cuenta Cuenta de AWS al crear la cuenta. Cuando se produce una actividad en AWS IoT Greengrass, esa actividad se registra en un CloudTrail evento junto con otros eventos de AWS servicio en el historial de eventos. Puede ver, buscar y descargar eventos recientes en su Cuenta de AWS. Para obtener más información, consulte Visualización de eventos con el historial de CloudTrail eventos.

Para tener un registro continuo de tus eventos Cuenta de AWS, incluidos los eventos para AWS IoT Greengrass ti, crea una ruta. Un rastro permite CloudTrail enviar archivos de registro a un bucket de S3. De forma predeterminada, al crear una ruta en la consola, la ruta se aplica a todos los Región de AWS s. La ruta registra los eventos de todas las regiones de la AWS partición y envía los archivos de registro al depósito de S3 que especifique. Además, puede configurar otros AWS servicios para analizar más a fondo los datos de eventos recopilados en los CloudTrail registros y actuar en función de ellos. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- CloudTrail servicios e integraciones compatibles
- Configuración de las notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de CloudTrail registro de varias regiones y recibir archivos de CloudTrail registro de varias cuentas

Todas AWS IoT Greengrass V2 las acciones se registran CloudTrail y se documentan en la referencia de la AWS IoT Greengrass V2 API. Por ejemplo, las llamadas a `CreateDeployment` y `CancelDeployment` las acciones generan entradas en los archivos de CloudTrail registro. `CreateComponentVersion`

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de usuario root o AWS Identity and Access Management (IAM).
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro AWS servicio.

Para obtener más información, consulte el elemento userIdentity de CloudTrail .

AWS IoT Greengrass eventos de datos en CloudTrail

Los eventos de datos proporcionan información sobre las operaciones de recursos realizadas en un recurso o dentro de un recurso (por ejemplo, obtener la versión de un componente o configurar una implementación). Se denominan también operaciones del plano de datos. Los eventos de datos suelen ser actividades de gran volumen. De forma predeterminada, CloudTrail no registra los eventos de datos. El historial de CloudTrail eventos no registra los eventos de datos.

Se aplican cargos adicionales a los eventos de datos. Para obtener más información sobre CloudTrail los precios, consulta AWS CloudTrail Precios.

Puede registrar eventos de datos para los tipos de AWS IoT Greengrass recursos mediante la CloudTrail consola o las operaciones de la CloudTrail API. AWS CLI La tabla de esta sección muestra los tipos de recursos disponibles para AWS IoT Greengrass.

- Para registrar eventos de datos mediante la CloudTrail consola, cree un almacén de datos de rutas o eventos para registrar eventos de datos, o actualice un banco de datos de seguimiento o evento existente para registrar eventos de datos.

  1. Para registrar eventos de datos, elija Eventos de datos.
  2. En la lista Tipo de evento de datos, elija el tipo de recurso en el que desea registrar los eventos de datos.

3. Elija la plantilla de selector de registro que desea usar. Puede registrar todos los eventos de datos del tipo de recurso, registre todos los eventos `readOnly`, registre todos los eventos `writeOnly` o cree una plantilla de selector de registro personalizada para filtrar por los campos `readOnly`, `eventName` y `resources.ARN`.

- Para registrar eventos de datos mediante el AWS CLI, configure el `--advanced-event-selectors` parámetro para que el `eventCategory` campo sea igual al valor del tipo de recurso `Data` y el `resources.type` campo igual al valor del tipo de recurso (consulte la tabla). Puede agregar condiciones para filtrar los valores de los campos `readOnly`, `eventName` y `resources.ARN`.
  - Para configurar el registro de seguimiento para eventos de datos de registro, ejecute el comando put-event-selectors. Para obtener más información, consulte Registro de eventos de datos para registros de seguimiento en la AWS CLI.
  - Para configurar un almacén de datos de eventos para registrar eventos de datos, ejecute el comando create-event-data-store para crear un almacén de datos de eventos nuevo o ejecute el comando update-event-data-store para actualizar uno existente. Para obtener más información, consulte Registro de eventos de datos para almacenes de datos de eventos con la AWS CLI.

En la siguiente tabla se enumeran los tipos de AWS IoT Greengrass recursos. La columna Tipo de evento de datos (consola) muestra el valor que se puede elegir en la lista de tipos de eventos de datos de la CloudTrail consola. La columna de valores resources.type muestra el `resources.type` valor, que se especificaría al configurar los selectores de eventos avanzados mediante o. AWS CLI CloudTrail APIs La CloudTrail columna Datos APIs registrados muestra las llamadas a la API registradas CloudTrail para el tipo de recurso.

Tipo de evento de datos (consola)	resources.type value	Datos APIs registrados en CloudTrail
Certificado IoT	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
Versión del componente IoT Greengrass	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates

Tipo de evento de datos (consola)	resources.type value	Datos APIs registrados en CloudTrail
Implementación de IoT Greengrass	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
Objeto de IoT	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

Puede configurar selectores de eventos avanzados para filtrar según los campos `eventName`, `readOnly` y `resources.ARN` y así registrar solo los eventos que son importantes para usted.

Añada un filtro `eventName` para incluir o excluir datos específicos APIs.

Para obtener más información acerca de estos campos, consulte AdvancedFieldSelector.

En los siguientes ejemplos, se muestra cómo configurar selectores avanzados con la AWS CLI. Sustituya *`TrailName`* y *`region`* con su propia información.

Example— Registro de eventos de datos para objetos de IoT

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log all thing data events",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
            ]
        }
]'
```

Example— Filtro de una API de un objeto de IoT específica

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
```

```
--advanced-event-selectors \
'[
    {
            "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
                { "Field": "eventName", "Equals": ["PutCertificateAuthorities"] }
            ]
    }
]'
```

Example— Registro de todos los eventos de datos de Greengrass

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
        "Name": "Log all certificate data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::IoT::Certificate"
                ]
            }
        ]
    },
    {
        "Name": "Log all component version data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
```

```
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::ComponentVersion”
                ]
            }
        ]
    },
    {
        “Name”: “Log all deployment version”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::Deployment”
                ]
            }
        ]
    },
    {
        “Name”: “Log all thing data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Thing”
                ]
            }
        ]
    }
]’
```

AWS IoT Greengrass eventos de gestión en CloudTrail

Los eventos de administración proporcionan información sobre las operaciones de administración que se realizan en los recursos de su AWS cuenta. Se denominan también operaciones del plano de control. De forma predeterminada, CloudTrail registra los eventos de administración.

AWS IoT Greengrass registra todas las operaciones del plano de AWS IoT Greengrass control como eventos de administración. Para obtener una lista de las operaciones del plano de AWS IoT Greengrass control en las que se AWS IoT Greengrass registra CloudTrail, consulte la referencia de la AWS IoT Greengrass API, versión 2.

Descripción de las entradas de los archivos de AWS IoT Greengrass V2 registro

Un registro es una configuración que permite la entrega de eventos como archivos de registro a un bucket de S3 que usted especifique. CloudTrail Los archivos de registro contienen una o más entradas de registro. Un evento representa una única solicitud desde cualquier origen. Incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un registro ordenado de las llamadas a las API públicas, por lo que no aparecen en ningún orden específico.

En el siguiente ejemplo, se muestra una entrada de CloudTrail registro que demuestra la `CreateDeployment` acción.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2021-01-06T02:38:05Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
```

```
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}
```


Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass

Los datos de telemetría del estado del sistema son datos de diagnóstico que pueden ayudarlo a monitorear el rendimiento de las operaciones críticas en sus dispositivos principales de Greengrass. Puede crear proyectos y aplicaciones para recuperar, analizar, transformar y generar informes sobre

los datos de telemetría de sus dispositivos periféricos. Los expertos de dominio, como los ingenieros de procesos, pueden utilizar estas aplicaciones para obtener información sobre el estado de la flota.

Puede utilizar los siguientes métodos para recopilar datos de telemetría de sus dispositivos principales de Greengrass:

- Componente emisor de telemetría de núcleo: el componente emisor de telemetría de núcleo (`aws.greengrass.telemetry.NucleusEmitter`) de un dispositivo principal de Greengrass publica los datos de telemetría en el tema `$local/greengrass/telemetry` de forma predeterminada. Puede utilizar los datos publicados en este tema para actuar de forma local en su dispositivo principal, incluso si su conectividad a la nube es limitada. Si lo desea, también puede configurar el componente para que publique los datos de telemetría en el tema de MQTT que prefiera. AWS IoT Core

  Debe implementar el componente emisor de núcleo en un dispositivo principal para publicar los datos de telemetría. La publicación de los datos de telemetría en el tema local no conlleva ningún costo. Sin embargo, el uso de un tema de MQTT para publicar datos en el Nube de AWS está sujeto a un precio.AWS IoT Core

  AWS IoT Greengrass proporciona varios componentes comunitarios para ayudarlo a analizar y visualizar los datos de telemetría localmente en su dispositivo principal mediante InfluxDB y Grafana. Estos componentes utilizan datos de telemetría del componente emisor de núcleo. Para obtener más información, consulte el archivo README para el componente publicador de InfluxDB.
- Agente de telemetría: el agente de telemetría de los dispositivos principales de Greengrass recopila datos de telemetría locales y los publica en Amazon sin necesidad de interacción con el cliente. EventBridge Los dispositivos principales publican los datos de telemetría haciendo el mejor esfuerzo. EventBridge Por ejemplo, es posible que los dispositivos principales no entreguen los datos de telemetría cuando están fuera de línea.

  La característica del agente de telemetría está habilitada de forma predeterminada en todos los dispositivos principales de Greengrass. Empezará a recibir datos automáticamente tan pronto como configure un dispositivo principal de Greengrass. Además de los costos de enlace de datos, la transferencia de datos desde el dispositivo principal AWS IoT Core es gratuita. Esto se debe a que el agente publica en un tema AWS reservado. Sin embargo, en función de su caso de uso, es posible que incurra en costos cuando reciba o procese los datos.

> **Note**
>
> Amazon EventBridge es un servicio de bus de eventos que puede utilizar para conectar sus aplicaciones con datos de diversas fuentes, como los dispositivos principales de Greengrass. Para obtener más información, consulta ¿Qué es Amazon EventBridge? en la Guía del EventBridge usuario de Amazon.

Para garantizar que el software AWS IoT Greengrass Core funcione correctamente, AWS IoT Greengrass utiliza los datos con fines de desarrollo y mejora de la calidad. Esta función también ayuda a informar sobre las capacidades perimetrales nuevas y mejoradas. AWS IoT Greengrass conserva los datos de telemetría durante un máximo de siete días.

En esta sección, se describe cómo configurar y utilizar el agente de telemetría. Para obtener información sobre la configuración del componente emisor de telemetría del núcleo, consulte Emisor de telemetría del núcleo.

Temas

- Métricas de telemetría
- Configuración de los ajustes del agente de telemetría
- Suscríbase a los datos de telemetría en EventBridge

Métricas de telemetría

En la siguiente tabla, se describen las métricas publicadas por el agente de telemetría.

Name	Description (Descripción)	
System (Sistema)		
`SystemMemUsage`	La cantidad de memoria que utilizan actualmente todas las aplicaciones del dispositivo principal de Greengrass, incluido el sistema operativo.	

Name	Description (Descripción)	
`CpuUsage`	La cantidad de CPU que utilizan actualmente todas las aplicaciones del dispositivo principal de Greengrass, incluido el sistema operativo.	
`TotalNumberOfFDs`	El número de descriptores de archivos almacenados por el sistema operativo del dispositivo principal de Greengrass. Un descriptor de archivo identifica exclusivamente un archivo abierto.	
Núcleo de Greengrass		
`NumberOfComponentsRunning`	El número de componentes que se ejecutan en el dispositivo principal de Greengrass.	
`NumberOfComponentsErrored`	El número de componentes que están en estado de error en el dispositivo principal de Greengrass.	
`NumberOfComponentsInstalled`	El número de componentes instalados en el dispositivo principal de Greengrass.	
`NumberOfComponentsStarting`	El número de componentes que se inician en el dispositivo principal de Greengrass.	
`NumberOfComponentsNew`	El número de componentes nuevos en el dispositivo principal de Greengrass.	

Name	Description (Descripción)	
`NumberOfComponents Stopping`	El número de componentes que se detienen en el dispositivo principal de Greengrass.	
`NumberOfComponents Finished`	El número de componentes terminados en el dispositivo principal de Greengrass.	
`NumberOfComponents Broken`	El número de componentes dañados en el dispositivo principal de Greengrass.	
`NumberOfComponents Stateless`	El número de componentes sin estado en el dispositivo principal de Greengrass.	
Autenticación del dispositivo de cliente: esta característica requiere la versión 2.4.0 o posterior del componente de autenticación del dispositivo de cliente.		
`VerifyClientDevice Identity.Success`	El número de veces que se ha comprobado que la identidad del dispositivo de cliente se ha realizado correctamente.	
`VerifyClientDevice Identity.Failure`	El número de veces que se verificó que se produjo un error en la identidad del dispositivo de cliente.	

Name	Description (Descripción)	
`AuthorizeClientDeviceActions.Success`	El número de veces que el dispositivo de cliente está autorizado a completar las acciones solicitadas.	
`AuthorizeClientDeviceActions.Failure`	El número de veces que el dispositivo de cliente no está autorizado a completar las acciones solicitadas.	
`GetClientDeviceAuthToken.Success`	El número de veces que el dispositivo de cliente se autentica correctamente.	
`GetClientDeviceAuthToken.Failure`	El número de veces que el dispositivo de cliente no se puede autenticar.	
`SubscribeToCertificateUpdates.Success`	El número de suscripciones correctas a actualizaciones de certificados.	
`SubscribeToCertificateUpdates.Failure`	El número de intentos fallidos de suscribirse a las actualizaciones de certificados.	
`ServiceError`	El número de errores internos no controlados en la autenticación del dispositivo de cliente.	
Administrador de flujos: esta característica requiere la versión 2.7.0 o posterior del componente núcleo de Greengrass.		

Name	Description (Descripción)	
`BytesAppended`	El número de bytes de datos anexos al administrador de flujos.	
`BytesUploadedToIoTAnalytics`	El número de bytes de datos que Stream Manager exporta a los canales. AWS IoT Analytics	
`BytesUploadedToKinesis`	El número de bytes de datos que el administrador de flujos exporta a los flujos de Amazon Kinesis Data Streams.	
`BytesUploadedToIoTSiteWise`	El número de bytes de datos que Stream Manager exporta a las propiedades de los activos AWS IoT SiteWise.	
`BytesUploadedToS3`	El número de bytes de datos que el administrador de flujos exporta a objetos de Amazon S3.	
Métricas de sistema: esta característica requiere la versión 2.15.0 o posterior del componente núcleo de Greengrass.		
`CPUArchitecture`	Arquitectura de la unidad central de procesamiento del dispositivo.	

Name	Description (Descripción)
`Family`	Familia de sistemas operativos de dispositivos (solo Windows).
`KernelVersion`	Versión kernel del dispositivo (solo Unix).
`KnowledgeBaseArticles`	Artículos de la base de conocimientos instalados en un dispositivo (solo para Windows).
`OSBuildMajor`	Número de compilación mayor de la versión del sistema operativo (solo Windows).
`OSBuildMinor`	Número de compilación menor de la versión del sistema operativo (solo Windows).
`OSName`	Nombre del sistema operativo del dispositivo.
`OSVersion`	Versión de marketing del sistema operativo del dispositivo.

Configuración de los ajustes del agente de telemetría

El agente de telemetría usa la configuración predeterminada:

- El agente de telemetría agrega los datos de telemetría cada hora.
- El agente de telemetría publica un mensaje de telemetría cada 24 horas.

El agente de telemetría publica los datos mediante el protocolo MQTT con un nivel de calidad de servicio (QoS) de 0, lo que significa que no confirma la entrega ni vuelve a intentar publicar los

intentos de publicación. Los mensajes de telemetría comparten una conexión MQTT con otros mensajes para las suscripciones destinadas para AWS IoT Core.

Además de los costes de enlace de datos, la transferencia de datos desde el núcleo AWS IoT Core es gratuita. Esto se debe a que el agente publica en un tema AWS reservado. Sin embargo, en función de su caso de uso, es posible que incurra en costos cuando reciba o procese los datos.

Puede habilitar o deshabilitar la característica de agente de telemetría para cada dispositivo principal de Greengrass. También puede configurar los intervalos durante los cuales el dispositivo principal agrega y publica datos. Para configurar la telemetría, personalice el parámetro de configuración de telemetría al implementar el componente de núcleo de Greengrass.

Suscríbase a los datos de telemetría en EventBridge

Puede crear reglas en Amazon EventBridge que definan cómo procesar los datos de telemetría publicados desde el agente de telemetría del dispositivo principal de Greengrass. Cuando EventBridge recibe los datos, invoca las acciones objetivo definidas en sus reglas. Por ejemplo, puede crear reglas de eventos que envíen notificaciones, almacenen información sobre eventos, adopten medidas correctivas o invoquen otros eventos.

Eventos de telemetría

Los eventos de telemetría utilizan el siguiente formato.

```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-11-30T20:45:53Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "ThingName": "MyGreengrassCore",
    "Schema": "2020-07-30",
    "ADP": [
      {
        "TS": 1602186483234,
        "NS": "SystemMetrics",
        "M": [
          {
```

```
            "N": "TotalNumberOfFDs",
            "Sum": 6447.0,
            "U": "Count"
          },
          {
            "N": "CpuUsage",
            "Sum": 15.458333333333332,
            "U": "Percent"
          },
          {
            "N": "SystemMemUsage",
            "Sum": 10201.0,
            "U": "Megabytes"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "GreengrassComponents",
        "M": [
          {
            "N": "NumberOfComponentsStopping",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStarting",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsBroken",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsFinished",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsInstalled",
            "Sum": 0.0,
            "U": "Count"
```

```
        },
        {
          "N": "NumberOfComponentsRunning",
          "Sum": 7.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsNew",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsErrored",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsStateless",
          "Sum": 0.0,
          "U": "Count"
        }
      ]
    },
    {
      "TS": 1602186483234,
      "NS": "aws.greengrass.ClientDeviceAuth",
      "M": [
        {
          "N": "VerifyClientDeviceIdentity.Success",
          "Sum": 3.0,
          "U": "Count"
        },
        {
          "N": "VerifyClientDeviceIdentity.Failure",
          "Sum": 1.0,
          "U": "Count"
        },
        {
          "N": "AuthorizeClientDeviceActions.Success",
          "Sum": 20.0,
          "U": "Count"
        },
        {
          "N": "AuthorizeClientDeviceActions.Failure",
```

```
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Success",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Failure",
            "Sum": 2.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Success",
            "Sum": 10.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "ServiceError",
            "Sum": 3.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.StreamManager",
        "M": [
          {
            "N": "BytesAppended",
            "Sum": 157745524.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTAnalytics",
            "Sum": 149012.0,
            "U": "Bytes"
          },
```

```
        {
          "N": "BytesUploadedToKinesis",
          "Sum": 12192.0,
          "U": "Bytes"
        },
        {
          "N": "BytesUploadedToIoTSiteWise",
          "Sum": 13321.0,
          "U": "Bytes"
        },
        {
          "N": "BytesUploadedToS3",
          "Sum": 12213.0,
          "U": "Bytes"
        }
      ]
    }
    {
      "TS": 1750104449426,
      "NS": "SystemMetrics",
      "M": [
        {
          "N": "KernelVersion",
          "Sum": 1,
          "U": "6.1.140-154.222.amzn2023.x86_64"
        },
        {
          "N": "OSVersion",
          "Sum": 1,
          "U": "2023.7.20250609"
        },
        {
          "N": "OSName",
          "Sum": 1,
          "U": "Amazon Linux"
        },
        {
          "N": "CPUArchitecture",
          "Sum": 1,
          "U": "Broadwell (Server)"
        },
      ]
    }
  ]
```

```
  }
}
```

La matriz ADP contiene una lista de puntos de datos agregados que tienen las siguientes propiedades:

`TS`

Marca temporal del momento en que se recopilaron los datos.

`NS`

Espacio de nombres de la métrica.

`M`

Lista de métricas Una métrica contiene las siguientes propiedades:

`N`

El nombre de la métrica.

`Sum`

La suma de los valores de la métrica en este evento de telemetría.

`U`

La unidad del valor métrico.

Para obtener más información acerca de cada métrica, consulte Métricas de telemetría.

Requisitos previos para crear reglas EventBridge

Antes de crear una EventBridge regla para AWS IoT Greengrass, debe hacer lo siguiente:

- Familiarícese con los eventos, las reglas y los objetivos en EventBridge.
- Cree y configure los objetivos invocados por sus EventBridge reglas. Las reglas pueden invocar muchos tipos de objetivos, como transmisiones de Amazon Kinesis AWS Lambda , funciones, temas de Amazon SNS y colas de Amazon SQS.

  Tu EventBridge regla y los objetivos asociados deben estar en el Región de AWS lugar donde creaste tus recursos de Greengrass. Para obtener más información, consulte Puntos de enlace y cuotas en la Referencia general de AWS.

Para obtener más información, consulta ¿Qué es Amazon EventBridge? y Primeros pasos con Amazon EventBridge en la Guía del EventBridge usuario de Amazon.

Cree una regla de eventos para obtener datos de telemetría (consola)

Siga los siguientes pasos Consola de administración de AWS para crear una EventBridge regla que reciba los datos de telemetría publicados por el dispositivo principal de Greengrass. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento. Para obtener más información, consulta Crear una EventBridge regla que se active en un evento desde un AWS recurso en la Guía del EventBridge usuario de Amazon.

1. Abre la EventBridgeconsola de Amazon y selecciona Crear regla.
2. En Name and description (Nombre y descripción), escriba un nombre y descripción para la regla.
3. En Define pattern (Definir patrón), configure el patrón de regla.
   a. Seleccione Event pattern.
   b. Elija Pre-defined pattern by service (Patrón predefinido por servicio).
   c. En Service provider (Proveedor de servicios), elija AWS.
   d. En Service name (Nombre del servicio), elija Greengrass.
   e. En Tipo de evento, seleccione Datos de telemetría de Greengrass.
4. En Select event bus (Seleccionar bus de evento), mantenga las opciones de bus de eventos predeterminadas.
5. En Select targets (Seleccionar destinos), configure su destino. El siguiente ejemplo usa una cola de Amazon SQS, pero puede configurar otros tipos de destinos.
   a. En Destino, elija la Cola SQS.
   b. En Cola*, seleccione la cola de destino.
6. En Tags - optional (Etiquetas - opcional), defina etiquetas para la regla o deje los campos vacíos.
7. Seleccione Crear.

Cree una regla de eventos para obtener datos de telemetría (CLI)

Siga los siguientes pasos AWS CLI para crear una EventBridge regla que reciba los datos de telemetría publicados por los dispositivos principales de Greengrass. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento.

1. Crear la regla.

   - *thing-name*Sustitúyala por el nombre del dispositivo principal.

   Linux or Unix

   ```
   aws events put-rule \
     --name MyGreengrassTelemetryEventRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   Windows Command Prompt (CMD)

   ```
   aws events put-rule ^
     --name MyGreengrassTelemetryEventRule ^
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   PowerShell

   ```
   aws events put-rule `
     --name MyGreengrassTelemetryEventRule `
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

   Las propiedades que se omiten en el patrón no se tienen en cuenta.

2. Agregue el tema como destino de la regla. El siguiente ejemplo usa Amazon SQS, pero puede configurar otros tipos de objetivos.

   - *queue-arn*Sustitúyalo por el ARN de la cola de Amazon SQS.

   Linux or Unix

   ```
   aws events put-targets \
     --rule MyGreengrassTelemetryEventRule \
     --targets "Id"="1","Arn"="queue-arn"
   ```

Windows Command Prompt (CMD)

```
aws events put-targets ^
  --rule MyGreengrassTelemetryEventRule ^
  --targets "Id"="1","Arn"="queue-arn"
```

PowerShell

```
aws events put-targets `
  --rule MyGreengrassTelemetryEventRule `
  --targets "Id"="1","Arn"="queue-arn"
```

> **Note**
>
> Para permitir que Amazon EventBridge invoque tu cola de destino, debes añadir a tu tema una política basada en recursos. Para obtener más información, consulte los permisos de Amazon SQS en la Guía EventBridge del usuario de Amazon.

Para obtener más información, consulta Eventos y patrones de eventos EventBridge en la Guía del EventBridge usuario de Amazon.

Reciba notificaciones sobre el estado de la implementación y de los componentes

Las reglas de EventBridge eventos de Amazon le proporcionan notificaciones sobre los cambios de estado de las implementaciones de Greengrass que reciben sus dispositivos y de los componentes instalados en sus dispositivos. EventBridge ofrece un flujo casi en tiempo real de los eventos del sistema que describe los cambios en AWS los recursos. AWS IoT Greengrass envía estos eventos haciendo EventBridge el mejor esfuerzo posible. Esto significa que AWS IoT Greengrass intenta enviar todos los eventos EventBridge pero, en algunos casos excepcionales, es posible que no se entregue un evento. Además, AWS IoT Greengrass puede enviar varias copias de un evento determinado, lo que significa que es posible que los oyentes del evento no reciban los eventos en el orden en que se produjeron.

> **Note**
>
> Amazon EventBridge es un servicio de bus de eventos que puede utilizar para conectar sus aplicaciones con datos de diversas fuentes, como los dispositivos principales de Greengrass y las notificaciones de despliegue y componentes. Para obtener más información, consulta ¿Qué es Amazon EventBridge? en la Guía del EventBridge usuario de Amazon.

Temas

- Evento de cambio de estado de una implementación
- Evento de cambio de estado del componente
- Requisitos previos para crear reglas EventBridge
- Configuración de las notificaciones de estado del dispositivo (consola)
- Configuración de las notificaciones de estado del dispositivo (CLI)
- Configuración de las notificaciones de estado del dispositivo (CloudFormation)
- Véase también

Evento de cambio de estado de una implementación

AWS IoT Greengrass emite un evento cuando una implementación entra en los siguientes estados:`FAILED`,`SUCCEEDED`, `COMPLETEDREJECTED`, y`CANCELED`. Puede crear una EventBridge regla que se aplique a todas las transiciones de estado o a los estados que especifique. Cuando una implementación entra en un estado que inicia una regla, EventBridge invoca las acciones objetivo definidas en la regla. Esto le permite enviar notificaciones, capturar información sobre el evento, tomar medidas correctivas o iniciar otros eventos en respuesta a un cambio de estado. Por ejemplo, puede crear reglas para los siguientes casos de uso:

- Iniciar operaciones posteriores a la implementación, como descargar recursos y enviar notificaciones al personal.
- Envíe notificaciones en caso de una implementación correcta o con error.
- Publicar métricas personalizadas sobre los eventos de implementación.

El evento de un cambio de estado de la implementación tiene el siguiente formato:

```
{
```

```
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Effective Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail":{
        "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED|REJECTED|CANCELED",
        "statusDetails": {
          "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR",
 "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
          "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
        },
        "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to
 download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for
 component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access
 Denied. Ensure the IAM role associated with the core device has a policy granting
 s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
 Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
luLCYANgq08RIH+x2H+hEKc=)"
     }
}
```

Puede crear reglas y eventos que informen sobre el estado de una implementación. Un evento se inicia cuando una implementación se completa como `FAILED`, `SUCCEEDED`, `COMPLETED`, `REJECTED` o `CANCELED`. Si la implementación falló en el dispositivo principal, recibirá una respuesta detallada que explica por qué la implementación falló. Para obtener más información sobre los códigos de error de la implementación, consulte Códigos de error de implementación detallados.

Estados de implementación

- `FAILED`. La implementación no se ha realizado correctamente
- `SUCCEEDED`: la implementación dirigida a un grupo de objetos se completó correctamente.
- `COMPLETED`: la implementación dirigida a un objeto se completó con éxito.
- `REJECTED`: la implementación se rechazó. Para obtener más información, consulte el campo `statusDetails`.
- `CANCELED`: la implementación se canceló por el usuario.

Es posible que los eventos se dupliquen o estén desordenados. Para determinar el orden de los eventos, utilice la propiedad `time`.

Para obtener una lista completa de los códigos de error en `errorStacks` y `errorTypes`, consulte Códigos de error de implementación detallados y Códigos de estado de componentes detallados.

Evento de cambio de estado del componente

Para AWS IoT Greengrass las versiones 2.12.2 y anteriores, Greengrass emite un evento cuando un componente entra en los siguientes estados: y. `ERRORED` `BROKEN` Para las versiones 2.12.3 y posteriores del núcleo de Greengrass, Greengrass emite un evento cuando un componente entra en los siguientes estados: `ERRORED`, `BROKEN`, `RUNNING` y `FINISHED`. Greengrass también emitirá un evento cuando se complete una implementación. Puede crear una EventBridge regla que se aplique a todas las transiciones de estado o a los estados que especifique. Cuando un componente instalado entra en un estado que inicia una regla, EventBridge invoca las acciones de destino definidas en la regla. Esto le permite enviar notificaciones, capturar información sobre el evento, tomar medidas correctivas o iniciar otros eventos en respuesta a un cambio de estado.

El evento de un cambio de estado del componente usa los siguientes formatos:

Greengrass nucleus v2.12.2 and earlier

<title>Estado del componente: ERRORED o BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
```

```
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

Greengrass nucleus v2.12.3 and later

<title>Estado del componente: ERRORED o BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

<title>Estado del componente: RUNNING o FINISHED</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
```

```
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "RUNNING|FINISHED",
            "lifecycleStateDetails": null
          }
        ]
    }
}
```

Puede crear reglas y eventos que informarán sobre el estado de un componente instalado. Se inicia un evento cuando un componente cambia de estado en el dispositivo. Recibirá una respuesta detallada en la que se explicará por qué un componente tiene errores o ha fallado. También recibirá un código de estado que indicará el motivo del fallo. Para obtener más información sobre los códigos del estado del componente, consulte Códigos de estado de componentes detallados.

Requisitos previos para crear reglas EventBridge

Antes de crear una EventBridge regla para AWS IoT Greengrass, haga lo siguiente:

- Familiarícese con los eventos, las reglas y los objetivos de EventBridge.
- Cree y configure los objetivos invocados por sus EventBridge reglas. Las reglas pueden invocar muchos tipos de destinos, entre los que se incluyen:
  - Amazon Simple Notification Service (Amazon SNS)
  - AWS Lambda funciones
  - Amazon Kinesis Video Streams
  - Colas de Amazon Simple Queue Service (Amazon SQS)

Para obtener más información, consulta ¿Qué es Amazon EventBridge? y Primeros pasos con Amazon EventBridge en la Guía del EventBridge usuario de Amazon.

Configuración de las notificaciones de estado del dispositivo (consola)

Siga los siguientes pasos para crear una EventBridge regla que publique un tema de Amazon SNS cuando cambie el estado de despliegue de un grupo. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento. Para obtener más información, consulta Crear una EventBridge regla que se active en un evento desde un AWS recurso en la Guía del EventBridge usuario de Amazon.

1. Abre la EventBridgeconsola de Amazon.
2. En el panel de navegación, seleccione Reglas.
3. Elija Creación de regla.
4. Escriba un nombre y una descripción para la regla.

   Una regla no puede tener el mismo nombre que otra regla de la misma región y del mismo bus de eventos.
5. En Bus de eventos, seleccione el bus de eventos que desea asociar a esta regla. Si desea que esta regla coincida con eventos procedentes de su cuenta, seleccione Bus de eventos predeterminado de AWS . Cuando un AWS servicio de tu cuenta emite un evento, siempre va al bus de eventos predeterminado de tu cuenta.
6. En Tipo de regla, elija Regla con un patrón de evento.
7. Seleccione Siguiente.
8. En Origen de eventos, seleccione (Eventos de AWS).
9. En Patrón de eventos, seleccione servicios AWS .
10. En Servicio de AWS , elija Greengrass.
11. En Tipo de evento, elija entre las siguientes opciones:

    - Para los eventos de implementación, elija Cambio de estado de la implementación efectiva de la versión 2 de Greengrass.
    - Para los eventos de componentes, elija Cambio de estado del componente instalado de la versión 2 de Greengrass.
12. Elija Siguiente.
13. En Tipos de destino (Tipos de destino), elija AWS service.
14. En Seleccionar destinos, configure su destino. En este ejemplo se utiliza un tema de Amazon SNS, pero se pueden configurar otros tipos de destino para enviar notificaciones.

a. En Destino, elija Tema de SNS.
b. En Topic (Tema), elija el tema de destino.
c. Elija Siguiente.

15. Elija Siguiente.
16. Revise los detalles de la regla y seleccione Creación de regla.

Configuración de las notificaciones de estado del dispositivo (CLI)

Siga los siguientes pasos para crear una EventBridge regla que publique un tema de Amazon SNS cuando se produzca un evento de cambio de estado de Greengrass. De este modo, los servidores web, las direcciones de correo electrónico y otros suscriptores del tema podrán responder al evento.

1. Crear la regla.

   - Para eventos de cambio de estado de una implementación.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Effective Deployment Status Change\"]}"
   ```

   - Para eventos de cambio de estado de un componente.

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Installed Component Status Change\"]}"
   ```

   Las propiedades que se omiten en el patrón no se tienen en cuenta.

2. Agregue el tema como destino de la regla.

   - *topic-arn*Sustitúyalo por el ARN de su tema de Amazon SNS.

   ```
   aws events put-targets \
     --rule TestRule \
     --targets "Id"="1","Arn"="topic-arn"
   ```

> **Note**
>
> Para permitir que Amazon llame EventBridge a tu tema objetivo, debes añadir a tu tema una política basada en recursos. Para obtener más información, consulte los permisos de Amazon SNS en la Guía EventBridge del usuario de Amazon.

Para obtener más información, consulta Eventos y patrones de eventos EventBridge en la Guía del EventBridge usuario de Amazon.

Configuración de las notificaciones de estado del dispositivo (CloudFormation)

Utilice CloudFormation plantillas para crear EventBridge reglas que envíen notificaciones sobre los cambios de estado para las implementaciones de su grupo de Greengrass. Para obtener más información, consulta la referencia de tipos de EventBridge recursos de Amazon en la Guía del AWS CloudFormation usuario.

Véase también

- Comprobación del estado de la implementación del dispositivo
- ¿Qué es Amazon EventBridge? en la Guía del EventBridge usuario de Amazon

Comprobación del estado del dispositivo principal de Greengrass

Los dispositivos principales de Greengrass informan del estado de sus componentes de software a. AWS IoT Greengrass Puede consultar el resumen del estado de cada dispositivo y comprobar el estado de cada componente en cada dispositivo.

Los dispositivos principales tienen los siguientes estados:

- `HEALTHY`— El software AWS IoT Greengrass principal y todos los componentes se ejecutan sin problemas en el dispositivo principal.
- `UNHEALTHY`— El software AWS IoT Greengrass principal o un componente se encuentra en un estado de error en el dispositivo principal.

> **Note**
>
> AWS IoT Greengrass depende de los dispositivos individuales para enviar actualizaciones de estado al Nube de AWS. Si el software AWS IoT Greengrass principal no se ejecuta en el dispositivo o si el dispositivo no está conectado al Nube de AWS, es posible que el estado informado de ese dispositivo no refleje su estado actual. La marca de tiempo del estado indica cuándo se actualizó por última vez el estado del dispositivo.
> Los dispositivos principales envían actualizaciones de estado en los siguientes momentos:
>
> - Cuando se inicia el software AWS IoT Greengrass Core
> - Cuando el dispositivo principal recibe una implementación del Nube de AWS
> - Para el núcleo de Greengrass 2.12.2 y versiones anteriores, el dispositivo principal envía actualizaciones de estado cuando el estado de cualquier componente del dispositivo principal pasa a ser `ERRORED` o `BROKEN`
> - Para el núcleo de Greengrass 2.12.3 y versiones posteriores, el dispositivo principal envía actualizaciones de estado cuando el estado de cualquier componente del dispositivo principal pasa a ser `ERRORED`, `BROKEN`, `RUNNING` o `FINISHED`
> - A un intervalo regular que puede configurar, que por defecto es de 24 horas
>
> En el AWS IoT Greengrass caso de Core v2.7.0 y versiones posteriores, el dispositivo principal envía actualizaciones de estado cuando se produce una implementación local y una implementación en la nube

Temas

- Comprobación del estado de un dispositivo principal
- Comprobación del estado de un grupo de dispositivos principales
- Comprobación del estado del componente del dispositivo principal

Comprobación del estado de un dispositivo principal

Puede comprobar el estado de dispositivos principales individuales.

Comprobación del estado de un dispositivo principal (AWS CLI)

- Ejecute el siguiente comando para recuperar el estado de un dispositivo. Reemplace *coreDeviceName* por el nombre del dispositivo principal que se va a consultar.

  ```
  aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
  ```

  La respuesta contiene información sobre el dispositivo principal, incluido su estado.

Comprobación del estado de un grupo de dispositivos principales

Puede comprobar el estado de un grupo de dispositivos principales (un grupo de objetos).

Comprobación del estado de un grupo de dispositivos (AWS CLI)

- Ejecute el siguiente comando para recuperar el estado de varios dispositivos principales. Reemplace el ARN del comando por el ARN del grupo de objetos que se va a consultar.

  ```
  aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-
  id:thinggroup/thingGroupName"
  ```

  La respuesta contiene la lista de dispositivos principales del grupo de objetos. Cada entrada de la lista contiene el estado del dispositivo principal.

Comprobación del estado del componente del dispositivo principal

Puede comprobar el estado, como el estado del ciclo de vida, de los componentes de software de un dispositivo principal. Para obtener más información sobre los estados del ciclo de vida del componente, consulte Desarrollo de componentes de AWS IoT Greengrass.

Comprobación del estado de los componentes de un dispositivo principal (AWS CLI)

- Ejecute el siguiente comando para recuperar el estado de los componentes de un dispositivo principal. Reemplace *coreDeviceName* por el nombre del dispositivo principal que se va a consultar.

  ```
  aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
  ```

La respuesta contiene la lista de componentes que se ejecutan en el dispositivo principal. Cada entrada de la lista contiene el estado del ciclo de vida del componente, incluido el estado actual de los datos y la última vez que el dispositivo principal de Greengrass envió un mensaje que contenía un determinado componente a la nube. La respuesta también incluirá el origen de la implementación más reciente que llevó el componente al dispositivo principal de Greengrass.

> **Note**
>
> Este comando recupera una lista paginada de los componentes que ejecuta un dispositivo principal de Greengrass. De forma predeterminada, esta lista no incluye los componentes que se implementan como dependencias de otros componentes. Puede incluir dependencias en la respuesta configurando el parámetro `topologyFilter` en `ALL`.

Ejecución de funciones de AWS Lambda

> **Note**
>
> AWS IoT Greengrass actualmente no admite esta característica en los dispositivos principales de Windows.

Puede importar funciones de AWS Lambda para que se ejecuten como componentes en los dispositivos principales de AWS IoT Greengrass. Es posible que desee hacerlo en las siguientes situaciones:

- Tiene un código de aplicación en las funciones de Lambda que desea implementar en los dispositivos principales.
- Tiene aplicaciones de AWS IoT Greengrass versión 1 que desea ejecutar en los dispositivos principales de AWS IoT Greengrass V2. Para obtener más información, consulte Paso 2: Crear e implementar AWS IoT Greengrass V2 componentes para migrar aplicaciones AWS IoT Greengrass V1.

Las funciones de Lambda incluyen dependencias de los siguientes componentes. No es necesario definir estos componentes como dependencias al importar la función. Si implementa un componente de función de Lambda, la implementación incluye estas dependencias del componente Lambda.

- El componente lanzador de Lambda (`aws.greengrass.LambdaLauncher`) gestiona los procesos y la configuración del entorno.
- El componente administrador de Lambda (`aws.greengrass.LambdaManager`) gestiona la comunicación y el escalado entre procesos.
- El componente de tiempos de ejecución de Lambda (`aws.greengrass.LambdaRuntimes`) proporciona artefactos para cada tiempo de ejecución de Lambda compatible.

Temas

- Requisitos
- Configuración del ciclo de vida de una función de Lambda
- Configuración de contenedores de funciones de Lambda

- Importación de una función de Lambda como componente (consola)
- Importación de una función de Lambda como componente (AWS CLI)

Requisitos

Sus dispositivos principales y las funciones de Lambda deben cumplir los siguientes requisitos para poder ejecutar las funciones en el software AWS IoT Greengrass Core:

- El dispositivo principal debe cumplir los requisitos para ejecutar las funciones de Lambda. Si desea que el dispositivo principal ejecute funciones de Lambda en contenedores, el dispositivo debe cumplir los requisitos para hacerlo. Para obtener más información, consulte Requisitos de la función de Lambda.
- Debe instalar los lenguajes de programación que utiliza la función de Lambda en sus dispositivos principales.

> **Tip**
>
> Puede crear un componente que instale el lenguaje de programación y, a continuación, especificar ese componente como una dependencia del componente de la función de Lambda. Greengrass es compatible con todas las versiones compatibles con Lambda de los tiempos de ejecución de Python, Node.js y Java. Greengrass no aplica ninguna restricción adicional a las versiones de tiempo de ejecución de Lambda obsoletas. Puede ejecutar funciones de Lambda que utilicen estos tiempos de ejecución obsoletos en AWS IoT Greengrass, pero no puede crearlas en AWS Lambda. Para obtener más información sobre la compatibilidad de AWS IoT Greengrass con los tiempos de ejecución de Lambda, consulte Ejecución de funciones de AWS Lambda.

Configuración del ciclo de vida de una función de Lambda

El ciclo de vida de la función de Lambda de Greengrass determina cuándo se inicia una función y cómo crea y utiliza contenedores. El ciclo de vida también determina cómo el software AWS IoT Greengrass Core retiene las variables y la lógica de preprocesamiento que están fuera del controlador de funciones.

AWS IoT Greengrass admite ciclos de vida bajo demanda (predeterminado) o de larga duración:

- Las funciones bajo demanda se inician cuando se invocan y se detienen cuando no quedan tareas que ejecutar. Cada invocación de la función crea un contenedor independiente, también entorno de pruebas, para procesar invocaciones, a menos que haya un contenedor disponible que se pueda reutilizar. Es posible que cualquiera de los contenedores procese los datos que envíe a la función.

  Es posible ejecutar en paralelo varias invocaciones de una función bajo demanda.

  No se conserva ninguna variable ni lógica de procesamiento previo que se defina fuera del controlador de la función cuando se crean nuevos contenedores.
- Las funciones de larga duración (o ancladas) se inician cuando se inicia el software AWS IoT Greengrass Core y se ejecutan en un solo contenedor. El mismo contenedor procesa todos los datos que se envían a la función.

  Se ponen en cola varias invocaciones hasta que el software AWS IoT Greengrass Core ejecuta las invocaciones anteriores.

  Las variables y lógica de procesamiento previo que se definen fuera del controlador de la función se conservan para cada invocación del controlador.

  Las funciones de Lambda de larga duración resultan útiles cuando necesita empezar a trabajar sin ninguna entrada inicial. Por ejemplo, una función de larga duración puede cargar y comenzar a procesar un modelo de machine learning para estar listo cuando la función reciba datos del dispositivo.

  > **Note**
  >
  > Recuerde que las funciones de larga duración tienen tiempos de espera que están asociados con invocaciones de su controlador. Si desea invocar el código que se ejecuta de forma indefinida, debe iniciarlo fuera del controlador. Asegúrese de que no haya código de bloqueo fuera del controlador que pudiera impedir la inicialización de la función. Estas funciones se ejecutan a menos que el software AWS IoT Greengrass Core se detenga, por ejemplo, durante una implementación o un reinicio. Estas funciones no se ejecutarán si la función encuentra una excepción no detectada, supera sus límites de memoria o entra en un estado de error, como el tiempo de espera del controlador.

Para obtener más información acerca de la reutilización de contenedores, consulte Comprender la reutilización de contenedores de AWS Lambda en el Blog de informática de AWS.

Configuración de contenedores de funciones de Lambda

De forma predeterminada, las funciones de Lambda se ejecutan dentro de un contenedor de AWS IoT Greengrass. Los contenedores de Greengrass proporcionan aislamiento entre sus funciones y el host. Este aislamiento aumenta la seguridad tanto del host como de las funciones del contenedor.

Le recomendamos que ejecute las funciones de Lambda en un contenedor de Greengrass a menos que su caso de uso requiera la ejecución sin contenedores. Al ejecutar las funciones de Lambda en un contenedor de Greengrass, tiene más control sobre la restricción de acceso a los recursos.

Puede ejecutar una función de Lambda sin contenedores en los siguientes casos:

- Desea ejecutar AWS IoT Greengrass en un dispositivo que no sea compatible con el modo en contenedores. Un ejemplo sería si quisiera usar una distribución especial de Linux o si tiene una versión anterior del núcleo que está desactualizada.
- Desea ejecutar su función de Lambda en otro entorno de contenedor con su propio OverlayFS, pero detecta conflictos de OverlayFS cuando la ejecuta en un contenedor de Greengrass.
- Necesita acceso a recursos locales con rutas que no se pueden determinar en el momento de la implementación o cuyas rutas pueden cambiar después de la implementación. Un ejemplo de este recurso sería un dispositivo conectable.
- Tiene una aplicación anterior que fue escrita como un proceso y encuentra problemas cuando la ejecuta en un contenedor de Greengrass.

Diferencias en la creación de contenedores

Creación de contenedores	Notas
Contenedor de Greengrass	• Todas las características de AWS IoT Greengrass están disponibles cuando ejecuta una función de Lambda en un contenedor de Greengrass. • Las funciones de Lambda que se ejecutan en un contenedor de Greengrass no tienen acceso al código implementado de otras

Creación de contenedores	Notas
	funciones de Lambda, incluso si se ejecutan con el mismo grupo de sistema. Es decir, las funciones de Lambda se ejecutan con mayor aislamiento de entre sí. • Dado que el software AWS IoT Greengrass Core ejecuta todos los procesos secundari os en el mismo contenedor que la función de Lambda, los procesos secundarios se detienen cuando se detiene la función de Lambda.

Sin contenedor	• Las siguientes características no están disponibles para funciones de Lambda no incluidas en contenedores:
	  • Límites de memoria de funciones de Lambda.
	  • Dispositivos locales y recursos de volumen. Debe acceder a estos recursos utilizando sus rutas de archivo en el dispositivo principal en lugar de hacerlo como recursos de la función de Lambda.
	• Si la función de Lambda que no está en un contenedor accede a un recurso de machine learning, debe identificar a un propietario del recurso y establecer permisos de acceso en el recurso, no en la función de Lambda.
	• La función de Lambda no incluida en contenedores tiene acceso de solo lectura al código implementado de otras funciones de Lambda que se están ejecutando con el mismo grupo de sistema.

Si se cambia la creación en contenedores para una función de Lambda durante la implementación, es posible que la función no funcione según lo esperado. Si había asignado recursos locales a la función de Lambda que ya no están disponibles con la nueva configuración de creación en contenedores, la implementación genera un error.

- Cuando se cambia una función de Lambda de la ejecución en un contenedor de Greengrass a la ejecución fuera de contenedores, se descartan los límites de memoria de la función. Debe acceder al sistema de archivos directamente en lugar de utilizar los recursos locales asociados. Debe eliminar todos los recursos adjuntos antes de implementar la función de Lambda.
- Al cambiar una función de Lambda de la ejecución sin creación de contenedores a la ejecución en un contenedor, la función de Lambda pierde el acceso directo al sistema de archivos. Debe definir un límite de memoria para cada función o aceptar la opción predeterminada de 16 MB. Puede establecer esta configuración para cada función de Lambda antes de la implementación.

Para cambiar la configuración en contenedores de un componente de la función de Lambda, defina el valor del parámetro de configuración `containerMode` en una de las siguientes opciones al implementar el componente.

- `NoContainer`: el componente no se ejecuta en un entorno de tiempo de ejecución aislado.
- `GreengrassContainer`: el componente se ejecuta en un entorno de tiempo de ejecución aislado dentro del contenedor de AWS IoT Greengrass.

Para obtener más información acerca de cómo implementar aplicaciones, consulte Implemente AWS IoT Greengrass componentes en los dispositivos y Actualización de las configuraciones de los componentes.

Importación de una función de Lambda como componente (consola)

Cuando utiliza la consola de AWS IoT Greengrass para crear un componente de una función de Lambda, importa una función de AWS Lambda y, a continuación, la configura para crear un componente que se ejecute en su dispositivo de Greengrass.

Antes de empezar, revise los requisitos para ejecutar las funciones de Lambda en los dispositivos de Greengrass.

Tareas

- Paso 1: Elegir una función de Lambda para importar
- Paso 2: Configurar los parámetros de la función de Lambda
- Paso 3: (Opcional) Especificar las plataformas compatibles con la función de Lambda
- Paso 4: (Opcional) Especificar las dependencias de los componentes para la función de Lambda
- Paso 5: (Opcional) Ejecutar la función de Lambda en un contenedor
- Paso 6: Crear el componente función de Lambda

Paso 1: Elegir una función de Lambda para importar

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, seleccione Crear componente.
3. En la página Crear componente, en Información del componente, elija Importar función de Lambda.
4. En la Función de Lambda, busque y elija la función de Lambda que desea importar.

   AWS IoT Greengrass crea el componente con el nombre de la función Lambda.
5. En la versión de la función de Lambda, elija la versión que desee importar. No puede elegir alias de Lambda como `$LATEST`.

   AWS IoT Greengrass crea el componente con la versión de la función Lambda como una versión semántica válida. Por ejemplo, si la versión de la función es `3`, la versión del componente se convierte en `3.0.0`.

Paso 2: Configurar los parámetros de la función de Lambda

En la página Crear componente, en Configuración de la función de Lambda, configure los siguientes parámetros para utilizarlos en la ejecución de la función de Lambda.

1. (Opcional) Agregue la lista de orígenes de eventos a los que se suscribe la función de Lambda para recibir mensajes de trabajo. Puede especificar las fuentes de eventos para suscribir esta función a los publish/subscribe mensajes locales y a los mensajes AWS IoT Core MQTT. Se llama a la función de Lambda cuando recibe un mensaje de un origen de evento.

> **Note**
>
> Para suscribir esta función a los mensajes de otras funciones o componentes de Lambda, implemente el componente enrutador de suscripciones heredado al implementar este componente de la función de Lambda. Al implementar el componente del enrutador de suscripciones heredado, especifique las suscripciones que utiliza la función de Lambda.

En Orígenes de evento, haga lo siguiente para agregar un origen del evento:

a. Para cada origen de evento que agregue, especifique las siguientes opciones:

   - Topic: el tema al que suscribirse a los mensajes.
   - Type: el tipo de origen de evento. Puede elegir entre las siguientes opciones:
     - Publicar/suscribirse de forma local: suscríbase a los mensajes locales. publish/subscribe

       Si usa el núcleo de Greengrass versión 2.6.0 o posterior y el administrador de Lambda versión 2.2.5 o posterior, puede usar los comodines de tema MQTT (+ y #) en el Topic cuando especifique este tipo.
     - AWS IoT Core MQTT: suscríbase a AWS IoT Core los mensajes de MQTT.

       Puede usar comodines de temas MQTT (+ y #) en el Topic cuando especifique este tipo.

b. Para agregar otro origen de evento, elija Agregar origen de evento y, a continuación, repita los pasos anteriores. Para eliminar un origen de evento, elija Eliminar que está ubicado junto al origen de evento que desea eliminar.

2. En Tiempo de espera (segundos), ingrese la cantidad máxima de tiempo en segundos que una función de Lambda no anclada puede ejecutarse antes de que se agote el tiempo de espera. El valor predeterminado es de 3 segundos.
3. En Anclado, elija si el componente de la función de Lambda está anclado. El valor predeterminado es true.

   - Una función Lambda anclada (o de larga duración) se inicia cuando se AWS IoT Greengrass inicia y sigue ejecutándose en su propio contenedor.

- Una función de Lambda no anclada (o bajo demanda) se inicia solo cuando recibe un elemento de trabajo y se cierra después de que se inactiva por un tiempo de inactividad máximo especificado. Si la función tiene varios elementos de trabajo, el software AWS IoT Greengrass Core crea varias instancias de la función.

4. (Opcional) En Parámetros adicionales, defina los siguientes parámetros de la función de Lambda.

   - Tiempo de espera del estado (segundos) : el intervalo en segundos, en el que el componente de la función de Lambda envía actualizaciones de estado al componente administrador de Lambda. Este parámetro solo se aplica a las funciones ancladas. El valor predeterminado es de 60 segundos.
   - Tamaño máximo de la cola: el tamaño máximo de la cola de mensajes para el componente de función de Lambda. El software AWS IoT Greengrass Core almacena los mensajes en una cola FIFO (primero en entrar, primero en salir) hasta que pueda ejecutar la función Lambda para consumir cada mensaje. El valor predeterminado es 1000 mensajes.
   - Máximo número de instancias: el número máximo de instancias que una función de Lambda no anclada puede ejecutar al mismo tiempo. El límite predeterminado es de 100 instancias.
   - Tiempo máximo de inactividad (segundos): cantidad máxima de tiempo en segundos que una función Lambda no anclada puede permanecer inactiva antes de que el software AWS IoT Greengrass principal detenga su proceso. El valor predeterminado es de 60 segundos.
   - Tipo de codificación: el tipo de carga útil que admite la función de Lambda. Puede elegir entre las siguientes opciones:
     - JSON
     - Binario

     El valor predeterminado es JSON.

5. (Opcional) Especifique la lista de argumentos de línea de comandos que se pasarán a la función de Lambda al ejecutarse.

   a. En Parámetros adicionales, argumentos de proceso, elija Agregar argumento.

   b. Para cada argumento que agregue, ingrese el argumento que desea pasar a la función.

   c. Para eliminar un argumento, elija Eliminar junto a la etiqueta que desee eliminar.

6. (Opcional) Especifique variables de entorno que están disponibles para la función de Lambda cuando se ejecuta. Las variables de entorno permiten almacenar y actualizar los valores de configuración sin necesidad de cambiar el código de la función.

a. En Parámetros adicionales, variables de entorno, elija Agregar variable de entorno.

b. Para cada variable de entorno que agregue, especifique las opciones siguientes:

- Clave: el nombre de la variable.
- Valor: el valor predeterminado de esta variable.

c. Para eliminar una variable de entorno, elija Eliminar que está ubicado junto a la variable de entorno que desea eliminar.

Paso 3: (Opcional) Especificar las plataformas compatibles con la función de Lambda

Todos los dispositivos principales tienen atributos de sistema operativo y arquitectura. Al implementar el componente de la función Lambda, el software AWS IoT Greengrass Core compara los valores de plataforma que especifique con los atributos de la plataforma en el dispositivo principal para determinar si la función Lambda es compatible con ese dispositivo.

> **Note**
>
> Puede especificar atributos de plataforma personalizados cuando implementa el componente núcleo de Greengrass en un dispositivo principal. Para obtener más información, consulte el parámetro de anulación de plataforma del componente núcleo de Greengrass.

En Configuración de la función de Lambda, Parámetros adicionales, Plataformas, haga lo siguiente para especificar las plataformas que admite esta función de Lambda.

1. Para cada plataforma, especifique las opciones siguientes:

   - Sistema operativo: el nombre del sistema operativo de la plataforma. Actualmente el único valor admitido es `linux`.
   - Arquitectura: la arquitectura del procesador de la plataforma. Los valores admitidos son:
     - `amd64`
     - `arm`
     - `aarch64`
     - `x86`

2. Para agregar otra plataforma, elija Agregar plataforma y repita el paso anterior. Para eliminar una plataforma compatible, elija Eliminar que está ubicado junto a la plataforma que desea eliminar.

Paso 4: (Opcional) Especificar las dependencias de los componentes para la función de Lambda

Las dependencias de los componentes identifican los componentes adicionales AWS proporcionados o los componentes personalizados que utiliza la función. Al implementar el componente de la función de Lambda, la implementación incluye estas dependencias para que la función se ejecute.

> **Important**
>
> Para importar una función Lambda que haya creado para ejecutarse en la AWS IoT Greengrass V1, debe definir las dependencias de los componentes individuales para las funciones que utiliza la función, como los secretos, las sombras locales y el administrador de flujos. Defina estos componentes como dependencias rígidas para que el componente de la función de Lambda se reinicie si la dependencia cambia de estado. Para obtener más información, consulte Cómo importar una función de Lambda V1.

En Configuración de la función de Lambda, Parámetros adicionales, Dependencias de componentes, complete los siguientes pasos para especificar las dependencias de los componentes de la función de Lambda.

1. Seleccione Agregar dependencia.
2. Para cada dependencia de componente que agregue, especifique las siguientes opciones:

   - Nombre del componente: el nombre del componente. Por ejemplo, introduzca **`aws.greengrass.StreamManager`** para incluir el componente administrador de flujos.
   - Requisito de versión: la restricción de versión semántica de estilo npm que identifica las versiones compatibles de esta dependencia del componente. Puede especificar una versión única o un rango de versiones. Por ejemplo, introduzca **`^1.0.0`** para especificar que esta función de Lambda depende de cualquier versión de la primera versión principal del componente administrador de flujos. Para obtener más información sobre las restricciones de la versión semántica, consulte la calculadora npm semver.
   - Tipo: el tipo de dependencia. Puede elegir entre las siguientes opciones:

- Restrictivo: el componente de la función de Lambda se reinicia si la dependencia cambia de estado. Esta es la selección predeterminada.
- Soft: el componente de la función de Lambda no se reinicia si la dependencia cambia de estado.

3. Para eliminar una dependencia de un componente, elija Eliminar junto a la dependencia del componente.

Paso 5: (Opcional) Ejecutar la función de Lambda en un contenedor

De forma predeterminada, las funciones Lambda se ejecutan en un entorno de ejecución aislado dentro del software AWS IoT Greengrass Core. También puede optar por ejecutar la función de Lambda como un proceso sin ningún tipo de aislamiento (es decir, en modo Sin contenedor).

En la Configuración de procesos de Linux, en el Modo de aislamiento, elija una de las siguientes opciones para seleccionar la contenerización de la función de Lambda:

- Contenedor de Greengrass: la función de Lambda se ejecuta en un contenedor. Esta es la selección predeterminada.
- Sin contenedor: la función de Lambda se ejecuta como un proceso sin ningún tipo de aislamiento.

Si ejecuta la función de Lambda en un contenedor, complete los siguientes pasos para establecer la configuración del proceso para la función de Lambda.

1. Configure la cantidad de memoria y los recursos del sistema, como los volúmenes y los dispositivos, que se pondrán a disposición del contenedor.

   En Parámetros de contenedor haga lo siguiente.

   a. En Tamaño de memoria, introduzca el tamaño de memoria que desee asignar al contenedor. Puede especificar el tamaño de la memoria en MB o KB.
   b. En Carpeta de sistema de solo lectura, elija si el contenedor puede leer o no información de la carpeta `/sys` del dispositivo. El valor predeterminado es false.

2. (Opcional) Configure los volúmenes locales a los que puede acceder la función de Lambda en contenedores. Al definir un volumen, el software AWS IoT Greengrass Core monta los archivos de origen en el destino dentro del contenedor.

   a. En Volúmenes, elija Agregar volumen.

b. Para cada volumen que agregue, especifique las siguientes opciones:

- Volumen físico: la ruta a la carpeta de origen en el dispositivo principal.
- Volumen lógico: la ruta a la carpeta de destino en el contenedor.
- Permission: (opcional) el permiso para acceder a la carpeta de origen desde el contenedor. Puede elegir entre las siguientes opciones:
  - Solo lectura: la función de Lambda tiene acceso de solo lectura a la carpeta de origen. Esta es la selección predeterminada.
  - Lectura y escritura: la función de Lambda tiene acceso de lectura y escritura a la carpeta de origen.
- Agregar propietario del grupo: (opcional) si se debe agregar o no el grupo de sistemas que ejecuta el componente de la función de Lambda como propietario de la carpeta de origen. El valor predeterminado es false.

c. Para eliminar una etiqueta, elija Eliminar junto al volumen que desee eliminar.

3. (Opcional) Configure los dispositivos del sistema local a los que puede acceder la función de Lambda en contenedores.

a. En Dispositivos, elija Agregar dispositivo.

b. Para cada dispositivo que agregue, especifique las siguientes opciones:

- Ruta de montaje: la ruta al dispositivo del sistema en el dispositivo principal.
- Permission: (opcional) el permiso para acceder al dispositivo del sistema desde el contenedor. Puede elegir entre las siguientes opciones:
  - Solo lectura: la función de Lambda tiene acceso de solo lectura al dispositivo del sistema. Esta es la selección predeterminada.
  - Lectura y escritura: la función de Lambda tiene acceso de lectura y escritura a la carpeta de origen.
- Agregar propietario del grupo: (opcional) si se debe agregar o no el grupo de sistemas que ejecuta el componente de la función de Lambda como propietario del dispositivo del sistema. El valor predeterminado es false.

Paso 6: Crear el componente función de Lambda

Después de configurar los ajustes del componente de la función de Lambda, elija Crear para terminar de crear el nuevo componente.

Para ejecutar la función de Lambda en su dispositivo principal, puede implementar el nuevo componente en sus dispositivos principales. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Importación de una función de Lambda como componente (AWS CLI)

Utilice la CreateComponentVersionoperación para crear componentes a partir de funciones Lambda. Cuando llame a esta operación, especifique `lambdaFunction` para importar una función de Lambda.

Tareas

- Paso 1: Definir la configuración de la función de Lambda
- Paso 2: Crear el componente función de Lambda

Paso 1: Definir la configuración de la función de Lambda

1. Cree un archivo llamado `lambda-function-component.json` y, a continuación, copie el siguiente objeto JSON en el archivo. Reemplace el `lambdaArn` por el ARN de la función de Lambda que se va a importar.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
     }
   }
   ```

   > **Important**
   >
   > Debe especificar un ARN que incluya la versión de la función que desea importar. No puede utilizar alias de versión como `$LATEST`.

2. (Opcional) Especifique el nombre (`componentName`) del componente. Si omite este parámetro, AWS IoT Greengrass crea el componente con el nombre de la función Lambda.

   ```
   {
     "lambdaFunction": {
   ```

```
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda"
  }
}
```

3. (Opcional) Especifique la versión (`componentVersion`) del componente. Si omite este parámetro, AWS IoT Greengrass crea el componente con la versión de la función Lambda como una versión semántica válida. Por ejemplo, si la versión de la función es `3`, la versión del componente se convierte en `3.0.0`.

> **Note**
>
> Cada versión de componente que cargue debe ser única. Asegúrese de cargar la versión del componente correcta, ya que no podrá editarla después de cargarla. AWS IoT Greengrass usa versiones semánticas para los componentes. Las versiones semánticas siguen un sistema de números de principal.secundario.parche. Por ejemplo, la versión `1.0.0` representa el primer lanzamiento principal de un componente. Para obtener más información, consulte la especificación semántica de la versión.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0"
  }
}
```

4. (Opcional) Especifique las plataformas compatibles con esta función de Lambda. Cada plataforma contiene un mapa de atributos que la identifican. Todos los dispositivos principales tienen atributos de sistema operativo (`os`) y arquitectura (`architecture`). El software AWS IoT Greengrass Core puede agregar otros atributos de plataforma. Puede especificar atributos de plataforma personalizados cuando implementa el componente núcleo de Greengrass en un dispositivo principal. Haga lo siguiente:

   a. Agregue una lista de plataformas (`componentPlatforms`) a la función de Lambda en `lambda-function-component.json`.

```
{
```

```
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [

    ]
  }
}
```

b. Agregue cada plataforma compatible a la lista. Cada plataforma tiene un `name` fácil de recordar para identificarla y un mapa de atributos. El siguiente ejemplo especifica que esta función es compatible con dispositivos x86 que ejecutan Linux.

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

Es posible que su `lambda-function-component.json` contenga un documento similar al ejemplo siguiente.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}
```

5. (Opcional) Especifique las dependencias de los componentes de la función de Lambda. Al implementar el componente de la función de Lambda, la implementación incluye estas dependencias para que la función se ejecute.

> **Important**
>
> Para importar una función Lambda que haya creado para ejecutarse en la AWS IoT Greengrass V1, debe definir las dependencias de los componentes individuales para las funciones que utiliza la función, como los secretos, las sombras locales y el administrador de flujos. Defina estos componentes como dependencias rígidas para que el componente de la función de Lambda se reinicie si la dependencia cambia de estado. Para obtener más información, consulte Cómo importar una función de Lambda V1.

Haga lo siguiente:

a. Agregue un mapa de las dependencias de los componentes (`componentDependencies`) a la función de Lambda en `lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {

    }
  }
}
```

b. Agregue cada dependencia de los componentes al mapa. Especifique el nombre del componente como clave y especifique un objeto con los siguientes parámetros:

- `versionRequirement`: la restricción de versión semántica de estilo npm que identifica las versiones compatibles de la dependencia del componente. Puede especificar una versión única o un rango de versiones. Para obtener más información sobre las restricciones de la versión semántica, consulte la calculadora npm semver.
- `dependencyType`: (opcional) el tipo de dependencia. Elija una de las siguientes opciones:
  - `SOFT`: el componente de la función de Lambda no se reinicia si la dependencia cambia de estado.
  - `HARD`: el componente de la función de Lambda no se reinicia si la dependencia cambia de estado.

  El valor predeterminado es `HARD`.

El siguiente ejemplo especifica que esta función de Lambda depende de cualquier versión de la primera versión principal del componente administrador de flujos. El componente de la función de Lambda se reinicia cuando el administrador de flujos se reinicia o se actualiza.

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

Es posible que su `lambda-function-component.json` contenga un documento similar al ejemplo siguiente.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
```

```
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}
```

6. (Opcional) Configure los parámetros de la función de Lambda que se utilizarán para ejecutar la función. Puede configurar opciones como las variables de entorno, los orígenes de eventos de los mensajes, los tiempos de espera y la configuración del contenedor. Haga lo siguiente:

   a. Agregue el objeto de parámetros de Lambda (`componentLambdaParameters`) a la función de Lambda en `lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {

    }
```

```
  }
}
```

b. (Opcional) Especifique los orígenes de eventos a los que se suscribe la función de Lambda para los mensajes de trabajo. Puede especificar las fuentes de eventos para suscribir esta función a los publish/subscribe mensajes locales y a los mensajes AWS IoT Core MQTT. Se llama a la función de Lambda cuando recibe un mensaje de un origen de evento.

> **Note**
>
> Para suscribir esta función a los mensajes de otras funciones o componentes de Lambda, implemente el componente enrutador de suscripciones heredado al implementar este componente de la función de Lambda. Al implementar el componente del enrutador de suscripciones heredado, especifique las suscripciones que utiliza la función de Lambda.

Haga lo siguiente:

i. Agregue la lista de orígenes de eventos (`eventSources`) a los parámetros de la función de Lambda.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
```

```
    },
    "componentLambdaParameters": {
      "eventSources": [

      ]
    }
  }
}
```

ii. Agregue cada origen de evento a la lista. Cada origen de evento tiene los siguientes parámetros:

- `topic`: el tema al que suscribirse a los mensajes.
- `type`: el tipo de origen de evento. Puede elegir entre las siguientes opciones:
  - PUB_SUB — Suscribirse a la mensajería de publicación/suscripción local.

    Si usa el núcleo de Greengrass versión 2.6.0 o posterior y el administrador de Lambda versión 2.2.5 o posterior, puede usar los comodines de tema MQTT (+ y #) en el `topic` cuando especifique este tipo.
  - IOT_CORE— Suscríbase a los mensajes de AWS IoT Core MQTT.

    Puede usar comodines de temas MQTT (+ y #) en el `topic` cuando especifique este tipo.

En el siguiente ejemplo, se suscribe a AWS IoT Core MQTT sobre temas que coinciden con el `hello/world/+` filtro de temas.

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

El `lambda-function-component.json` puede tener un aspecto similar al siguiente ejemplo.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
```

```
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
    }
  }
}
```

c. (Opcional) Especifique cualquiera de los siguientes parámetros en el objeto de parámetros de la función de Lambda:

- `environmentVariables`: el mapa de variables de entorno que están disponibles para la función de Lambda cuando se ejecuta.
- `execArgs`: la lista de argumentos para pasar a la función de Lambda cuando se ejecuta.
- `inputPayloadEncodingType`: el tipo de carga útil que admite la función de Lambda. Puede elegir entre las siguientes opciones:
  - `json`
  - `binary`

  Valor predeterminado: `json`
- `pinned`: si la función de Lambda está anclada o no. El valor predeterminado es `true`.

- Una función Lambda anclada (o de larga duración) se inicia cuando se AWS IoT Greengrass inicia y sigue ejecutándose en su propio contenedor.
- Una función de Lambda no anclada (o bajo demanda) se inicia solo cuando recibe un elemento de trabajo y se cierra después de que se inactiva por un tiempo de inactividad máximo especificado. Si la función tiene varios elementos de trabajo, el software AWS IoT Greengrass Core crea varias instancias de la función.

Se utiliza `maxIdleTimeInSeconds` para establecer el tiempo máximo de inactividad de la función.

- `timeoutInSeconds`: la cantidad máxima de tiempo en segundos que una función de Lambda puede ejecutarse antes de que se agote. El valor predeterminado es de 3 segundos.
- `statusTimeoutInSeconds`: intervalo en segundos en el que un componente de función de Lambda envía actualizaciones de estado al componente del administrador de Lambda. Este parámetro solo se aplica a las funciones ancladas. El valor predeterminado es de 60 segundos.
- `maxIdleTimeInSeconds`— El tiempo máximo en segundos que una función Lambda no anclada puede permanecer inactiva antes de que el software AWS IoT Greengrass Core detenga su proceso. El valor predeterminado es de 60 segundos.
- `maxInstancesCount`: el número máximo de instancias que una función de Lambda no anclada puede ejecutar al mismo tiempo. El límite predeterminado es de 100 instancias.
- `maxQueueSize`: tamaño máximo de la cola de mensajes para el componente de función de Lambda. El software AWS IoT Greengrass Core almacena los mensajes en una cola FIFO (first-in-first-out) hasta que pueda ejecutar la función Lambda para consumir cada mensaje. El valor predeterminado es 1000 mensajes.

Es posible que su `lambda-function-component.json` contenga un documento similar al ejemplo siguiente.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
```

```
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500
    }
  }
}
```

d. (Opcional) Configure los ajustes del contenedor para la función de Lambda. De forma predeterminada, las funciones Lambda se ejecutan en un entorno de ejecución aislado dentro del software AWS IoT Greengrass Core. También puede optar por ejecutar la función de Lambda como un proceso sin ningún tipo de aislamiento. Si ejecuta la función de Lambda en un contenedor, configura el tamaño de memoria del contenedor y los recursos del sistema disponibles para la función de Lambda. Haga lo siguiente:

i. Agregue el objeto de parámetros de proceso de Linux (`linuxProcessParams`) al objeto de parámetros de Lambda en `lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
```

```
      "maxQueueSize": 500,
      "linuxProcessParams": {

      }
    }
  }
}
```

ii. (Opcional) Especifique si la función de Lambda se ejecuta o no en un contenedor. Agregue el parámetro `isolationMode` al objeto de parámetros del proceso y elija una de las siguientes opciones:

- `GreengrassContainer`: la función de Lambda se ejecuta en un contenedor.
- `NoContainer`: la función de Lambda se ejecuta como un proceso sin ningún tipo de aislamiento.

El valor predeterminado es `GreengrassContainer`.

iii. (Opcional) Si ejecuta la función de Lambda en un contenedor, puede configurar la cantidad de memoria y los recursos del sistema, como los volúmenes y los dispositivos, que se pondrán a disposición del contenedor. Haga lo siguiente:

A. Agregue el objeto de parámetros del contenedor (`containerParams`) al objeto de parámetros de proceso de Linux en `lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
```

```
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {

        }
      }
    }
  }
}
```

B. (Opcional) Agregue el parámetro `memorySizeInKB` para especificar el tamaño de memoria del contenedor. El valor predeterminado es de 16 384 KB (16 MB).

C. (Opcional) Agregue el parámetro `mountROSysfs` para especificar si el contenedor puede leer o no información de la carpeta `/sys` del dispositivo. El valor predeterminado es `false`.

D. (Opcional) Configure los volúmenes locales a los que puede acceder la función de Lambda en contenedores. Al definir un volumen, el software AWS IoT Greengrass Core monta los archivos de origen en el destino dentro del contenedor. Haga lo siguiente:

I. Agregue la lista de volúmenes (`volumes`) a los parámetros del contenedor.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
```

```
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [

          ]
        }
      }
    }
  }
}
```

II. Agregue cada volumen a la lista. Cada volumen tiene los siguientes parámetros:

- `sourcePath`: la ruta a la carpeta de origen en el dispositivo principal.
- `destinationPath`: la ruta a la carpeta de destino en el contenedor.
- `permission`: (opcional) el permiso para acceder a la carpeta de origen desde el contenedor. Puede elegir entre las siguientes opciones:
  - `ro`: la función de Lambda tiene acceso de solo lectura a la carpeta de origen.
  - `rw`: la función de Lambda tiene acceso de lectura y escritura a la carpeta de origen.

  El valor predeterminado es `ro`.
- `addGroupOwner`: (opcional) si desea agregar o no el grupo de sistemas que ejecuta el componente de la función de Lambda como propietario de la carpeta de origen. El valor predeterminado es `false`.

Es posible que su `lambda-function-component.json` contenga un documento similar al ejemplo siguiente.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
```

```
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
```

```
            "addGroupOwner": true
          }
        ]
      }
    }
  }
}
```

E. (Opcional) Configure los dispositivos del sistema local a los que puede acceder la función de Lambda en contenedores. Haga lo siguiente:

I. Agregue la lista de dispositivos de sistema (`devices`) a los parámetros del contenedor.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
```

```
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [

          ]
        }
      }
    }
  }
}
```

II. Agregue cada dispositivo del sistema a la lista. Cada dispositivo del sistema tiene los siguientes parámetros:

- `path`: la ruta al dispositivo del sistema en el dispositivo principal.
- `permission`: (opcional) el permiso para acceder al dispositivo del sistema desde el contenedor. Puede elegir entre las siguientes opciones:
  - `ro`: la función de Lambda tiene acceso de solo lectura al dispositivo del sistema.

  - `rw`: la función de Lambda tiene acceso de lectura y escritura al dispositivo del sistema.

  El valor predeterminado es `ro`.

- `addGroupOwner`: (opcional) si desea agregar o no el grupo de sistemas que ejecuta el componente de la función de Lambda como propietario del dispositivo del sistema. El valor predeterminado es `false`.

Es posible que su `lambda-function-component.json` contenga un documento similar al ejemplo siguiente.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
```

```
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [
            {
              "path": "/dev/sda3",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

7. (Opcional) Agregue etiquetas (`tags`) para el componente. Para obtener más información, consulte Etiquete sus AWS IoT Greengrass Version 2 recursos.

Paso 2: Crear el componente función de Lambda

1. Ejecute el siguiente comando para crear el componente de la función de Lambda desde `lambda-function-component.json`.

   ```
   aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
   ```

   Si la solicitud se realiza con éxito, la respuesta de es similar a la del siguiente ejemplo.

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1
     "componentName": "com.example.HelloWorldLambda",
     "componentVersion": "1.0.0",
     "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
     "status": {
       "componentState": "REQUESTED",
       "message": "NONE",
       "errors": {}
     }
   }
   ```

   Copie el `arn` de la respuesta para comprobar el estado del componente en el paso siguiente.

2. Al crear un componente, su estado es `REQUESTED`. A continuación, AWS IoT Greengrass valida que el componente se pueda implementar. Puede ejecutar el siguiente comando para consultar el estado del componente y comprobar que el componente se puede implementar. Sustituya `arn` por el ARN del paso anterior.

   ```
   aws greengrassv2 describe-component \
     --arn "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0"
   ```

   Si el componente se valida, la respuesta indica que el estado del componente es `DEPLOYABLE`.

   ```
   {
     "arn": "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0",
     "componentName": "com.example.HelloWorldLambda",
   ```

```
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
}
```

Una vez que el componente está `DEPLOYABLE`, puede implementar la función de Lambda en los dispositivos principales. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core

Los componentes que se ejecutan en su dispositivo principal pueden utilizar la biblioteca de comunicación entre procesos (IPC) del AWS IoT Greengrass núcleo SDK para dispositivos con AWS IoT para comunicarse con el AWS IoT Greengrass núcleo y otros componentes de Greengrass. Para desarrollar y ejecutar componentes personalizados que utilicen la IPC, debe utilizarla SDK para dispositivos con AWS IoT para conectarse al servicio AWS IoT Greengrass Core IPC y realizar las operaciones de IPC.

La interfaz de IPC admite dos tipos de operaciones:

- Solicitud/respuesta

  Los componentes envían una solicitud al servicio de IPC y reciben una respuesta que contiene el resultado de la solicitud.
- Suscripción

  Los componentes envían una solicitud de suscripción al servicio de IPC y esperan recibir un flujo de mensajes de eventos como respuesta. Los componentes proporcionan un controlador de suscripciones que gestiona los mensajes de eventos, los errores y el cierre del flujo. SDK para dispositivos con AWS IoT Incluye una interfaz de controlador con la respuesta y los tipos de eventos correctos para cada operación de IPC. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Temas

- Versiones de cliente de IPC
- Compatible con SDKs la comunicación entre procesos
- Conéctese al AWS IoT Greengrass servicio Core IPC
- Autorización de los componentes para realizar operaciones de IPC
- Suscripción a los flujos de eventos de IPC
- Prácticas recomendadas de IPC
- Publicar/suscribir mensajes locales

- Publicar/suscribir mensajes MQTT AWS IoT Core
- Interacción con el ciclo de vida del componente
- Interacción con la configuración de componentes
- Recupere valores secretos
- Interactúe con las sombras locales
- Administre las implementaciones y los componentes locales
- Autenticación y autorización de los dispositivos de cliente

Versiones de cliente de IPC

En versiones posteriores de Java y Python SDKs, AWS IoT Greengrass proporciona una versión mejorada del cliente IPC, denominada cliente IPC V2. Cliente de IPC V2:

- Reduce la cantidad de código que debe escribirse para utilizar las operaciones de IPC y ayuda a evitar errores habituales que pueden producirse con el cliente de IPC V1.
- Realiza devoluciones de llamadas a los controladores de suscripciones en un subproceso independiente, por lo que ahora puede ejecutar código de bloqueo, incluidas las llamadas a funciones de IPC adicionales, en las devoluciones de llamadas a los controladores de suscripciones. El cliente de IPC V1 utiliza el mismo subproceso para comunicarse con el servidor de IPC y para llamar al controlador de suscripciones.
- Permite llamar a las operaciones de suscripción mediante expresiones Lambda (Java) o funciones (Python). El cliente de IPC V1 requiere que defina las clases de controladores de suscripciones.
- Proporciona versiones sincrónicas y asincrónicas de cada operación de IPC. El cliente de IPC V1 proporciona solo versiones asíncronas de cada operación.

Recomendamos utilizar el cliente de IPC V2 para aprovechar estas mejoras. Sin embargo, muchos ejemplos de esta documentación y de algunos contenidos en línea muestran únicamente cómo utilizar el cliente de IPC V1. Puede utilizar los siguientes ejemplos y tutoriales para ver ejemplos de componentes que utilizan el cliente de IPC V2:

- PublishToTopicejemplos
- SubscribeToTopicejemplos
- Tutorial: Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes

- Tutorial: interactúe con dispositivos IoT locales a través de MQTT

Actualmente, la versión 2 SDK para dispositivos con AWS IoT para C++ solo admite el cliente IPC V1.

Compatible con SDKs la comunicación entre procesos

Las bibliotecas AWS IoT Greengrass Core IPC se incluyen en las siguientes SDK para dispositivos con AWS IoT versiones.

SDK	Versión mínima	De uso
SDK para dispositivos con AWS IoT para Java v2	Versión 1.6.0	Consulte Úselo SDK para dispositivos con AWS IoT para Java v2 (cliente IPC V2)
SDK para dispositivos con AWS IoT para Python v2	Versión 1.9.0	Consulte Uso SDK para dispositivos con AWS IoT para Python v2 (cliente IPC V2)
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0	Consulte Úselo SDK para dispositivos con AWS IoT para C++ v2
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	Consulte Úselo SDK para dispositivos con AWS IoT para la JavaScript versión 2 (cliente IPC V1)

Conéctese al AWS IoT Greengrass servicio Core IPC

Para utilizar la comunicación entre procesos en su componente personalizado, debe crear una conexión a un socket de servidor IPC que ejecute el software AWS IoT Greengrass Core. Realice

las siguientes tareas para descargarlo y usarlo SDK para dispositivos con AWS IoT en el idioma que prefiera.

Úselo SDK para dispositivos con AWS IoT para Java v2 (cliente IPC V2)

Para usar la versión 2 SDK para dispositivos con AWS IoT para Java (cliente IPC V2)

1. Descargue el SDK para dispositivos con AWS IoT para Java v2 (versión 1.6.0 o posterior).
2. Tome alguna de las siguientes medidas para ejecutar el código personalizado en su componente:
   - Cree el componente como un archivo JAR que incluya el SDK para dispositivos con AWS IoT archivo JAR y ejecútelo en la receta del componente.
   - Defina el SDK para dispositivos con AWS IoT JAR como un artefacto de componente y añada ese artefacto a la ruta de clases cuando ejecute la aplicación en la receta de su componente.
3. Utilice el siguiente código para crear el cliente de IPC.

```
try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
}
```

Uso SDK para dispositivos con AWS IoT para Python v2 (cliente IPC V2)

Para usar SDK para dispositivos con AWS IoT para Python v2 (cliente IPC V2)

1. Descargue el SDK para dispositivos con AWS IoT para Python (versión 1.9.0 o posterior).
2. Agregue los pasos de instalación del SDK al ciclo de vida de instalación en la receta de su componente.
3. Cree una conexión con el servicio AWS IoT Greengrass Core IPC. Utilice el siguiente código para crear el cliente de IPC.

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

try:
```

```
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Úselo SDK para dispositivos con AWS IoT para C++ v2

Para compilar la SDK para dispositivos con AWS IoT versión 2 para C++, un dispositivo debe tener las siguientes herramientas:

- C++ 11 o posterior
- CMake 3.1 o posterior
- Uno de los siguientes copiladores:
  - GCC 4.8 o posterior
  - Clang 3.9 o posterior
  - MSVC 2015 o posterior

Para usar la versión 2 SDK para dispositivos con AWS IoT para C++

1. Descargue el SDK para dispositivos con AWS IoT para C++ v2 (versión 1.17.0 o posterior).
2. Siga las instrucciones de instalación del archivo README para compilar la versión 2 SDK para dispositivos con AWS IoT para C++ a partir del código fuente.
3. En la herramienta de compilación de C++, vincule la biblioteca de IPC de Greengrass, `AWS::GreengrassIpc-cpp`, que creó en el paso anterior. El siguiente `CMakeLists.txt` ejemplo vincula la biblioteca IPC de Greengrass a un proyecto con el que se crea. CMake

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
        "*.h"
        "*.cpp"
        )
add_executable(${PROJECT_NAME} ${MAIN_SRC})

set_target_properties(${PROJECT_NAME} PROPERTIES
```

```
        LINKER_LANGUAGE CXX
        CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
```

4. En el código del componente, cree una conexión al servicio AWS IoT Greengrass Core IPC para crear un cliente IPC (). `Aws::Greengrass::GreengrassCoreIpcClient` Debe definir un controlador del ciclo de vida de las conexiones de IPC que gestione los eventos de conexión, desconexión y error de IPC. El siguiente ejemplo crea un cliente de IPC y un controlador del ciclo de vida de las conexiones de IPC que se imprimen cuando el cliente de IPC se conecta, se desconecta y detecta errores.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
 std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
```

```
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    // Use the IPC client to create an operation request.

    // Activate the operation request.
    auto activate = operation.Activate(request, nullptr);
    activate.wait();

    // Wait for Greengrass Core to respond to the request.
    auto responseFuture = operation.GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
        exit(-1);
    }

    // Check the result of the request.
    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }
```

```
    return 0;
}
```

5. Para ejecutar el código personalizado en el componente, cree el código como un artefacto binario y ejecute el artefacto binario en la receta del componente. Establezca el `Execute` permiso del artefacto para `OWNER` permitir que el software AWS IoT Greengrass principal ejecute el artefacto binario.

   La sección `Manifests` de la receta de su componente podría parecerse al siguiente ejemplo.

   JSON

   ```
   {
     ...
     "Manifests": [
       {
         "Lifecycle": {
           "Run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
         },
         "Artifacts": [
           {
             "URI": "s3://amzn-s3-demo-bucket/artifacts/
   com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
             "Permission": {
               "Execute": "OWNER"
             }
           }
         ]
       }
     ]
   }
   ```

   YAML

   ```
   ...
   Manifests:
     - Lifecycle:
         Run: {artifacts:path}/greengrassv2_pubsub_subscriber
       Artifacts:
         - URI: s3://amzn-s3-demo-bucket/artifacts/
   com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
           Permission:
   ```

```
            Execute: OWNER
```

Úselo SDK para dispositivos con AWS IoT para la JavaScript versión 2 (cliente IPC V1)

Para compilar la JavaScript versión 2 SDK para dispositivos con AWS IoT para usarla con Nodejs, un dispositivo debe tener las siguientes herramientas:

- Node.JS 10.0 o posterior
  - Ejecute `node -v` para comprobar la versión de Node.
- CMake 3.1 o posterior

Para usar la SDK para dispositivos con AWS IoT JavaScript versión 2 (cliente IPC V1)

1. Descargue la JavaScript versión SDK para dispositivos con AWS IoT para la versión 2 (v1.12.10 o posterior).
2. Siga las instrucciones de instalación del archivo README para compilar la versión para la versión 2 a partir del código SDK para dispositivos con AWS IoT fuente JavaScript.
3. Cree una conexión al servicio AWS IoT Greengrass Core IPC. Realice los siguientes pasos para crear el cliente de IPC y establecer una conexión.
4. Utilice el siguiente código para crear el cliente de IPC.

   ```
   import * as greengrascoreipc from 'aws-iot-device-sdk-v2';

   let client = greengrascoreipc.createClient();
   ```

5. Utilice el siguiente código para establecer una conexión entre el componente y el núcleo de Greengrass.

   ```
   await client.connect();
   ```

Autorización de los componentes para realizar operaciones de IPC

Para permitir que sus componentes personalizados utilicen algunas operaciones de IPC, debe definir políticas de autorización que permitan al componente realizar la operación en determinados

recursos. Cada política de autorización define una lista de operaciones y una lista de recursos que la política permite. Por ejemplo, el servicio IPC de publish/subscribe mensajería define las operaciones de publicación y suscripción de los recursos temáticos. Puede utilizar el comodín * para permitir el acceso a todas las operaciones o a todos los recursos.

Las políticas de autorización se definen con el parámetro de configuración `accessControl`, que se puede establecer en la receta del componente o al implementar el componente. El objeto `accessControl` asigna los identificadores del servicio de IPC a listas de políticas de autorización. Puede definir varias políticas de autorización para cada servicio de IPC a fin de controlar el acceso. Cada política de autorización tiene un identificador de política, que debe ser único entre todos los componentes.

> **Tip**
>
> Para crear una política única IDs, puede combinar el nombre del componente, el nombre del servicio de IPC y un contador. Por ejemplo, un componente denominado `com.example.HelloWorld` podría definir dos políticas de publish/subscribe autorización con lo siguiente: IDs
>
> - `com.example.HelloWorld:pubsub:1`
> - `com.example.HelloWorld:pubsub:2`

Las políticas de autorización utilizan el siguiente formato. Este objeto es el parámetro de configuración `accessControl`.

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
```

```
    }
  }
}
```

YAML

```
IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
      - resource1
      - resource2
```

Comodines en las políticas de autorización

Puede utilizar el comodín * en el elemento `resources` de las políticas de autorización del IPC para permitir el acceso a varios recursos en una única política de autorización.

- En todas las versiones del núcleo de Greengrass, puede especificar un solo carácter * como recurso para permitir el acceso a todos los recursos.
- En el núcleo de Greengrass versión 2.6.0 y versiones posteriores, puede especificar el carácter * de un recurso para que coincida con cualquier combinación de caracteres. Por ejemplo, puede especificar `factory/1/devices/Thermostat*/status` para permitir el acceso a un tema de estado para todos los dispositivos de termostato de una fábrica, donde el nombre de cada dispositivo comience con `Thermostat`.

Al definir políticas de autorización para el servicio IPC de AWS IoT Core MQTT, también puede utilizar los caracteres comodín (+y#) de MQTT para hacer coincidir varios recursos. Para obtener más información, consulte los caracteres comodín de MQTT en las políticas de autorización de IPC de MQTT. AWS IoT Core

Variables de receta en las políticas de autorización

Si usa Greengrass nucleus v2.6.0 o posterior y establece la opción de interpolateComponentConfigurationconfiguración del núcleo de Greengrass en`true`, puede usar

la variable de receta en las políticas de autorización. `{iot:thingName}` Cuando necesite una política de autorización que incluya el nombre del dispositivo principal, como en el caso de temas de MQTT o sombras de dispositivo, puede utilizar esta variable de receta para configurar una política de autorización única para un grupo de dispositivos principales. Por ejemplo, puede permitir que un componente acceda al siguiente recurso para realizar operaciones de IPC de sombra.

```
$aws/things/{iot:thingName}/shadow/
```

Caracteres especiales en las políticas de autorización

Para especificar un literal * o un carácter ? en una política de autorización, debe utilizar una secuencia de escape. Las siguientes secuencias de escape indican al software AWS IoT Greengrass Core que utilice el valor literal en lugar del significado especial del carácter. Por ejemplo, el carácter * es un comodín que coincide con cualquier combinación de caracteres.

Carácter literal	Secuencia de escape	Notas
*	${*}	
?	${?}	AWS IoT Greengrass actualmente no admite el ? comodín, que coincide con cualquier carácter individual.
$	${$}	Use esta secuencia de escape para hacer coincidir un recurso que contenga ${. Por ejemplo, para que coincida con un recurso denominado ${resourceName}, debe especificar ${${}{resourceName}. De lo contrario, para que coincida con un recurso que contiene $, puede usar un literal $, por ejemplo, para permitir el acceso a un tema que comience por $aws.

Ejemplos de políticas de autorización

Puede consultar los siguientes ejemplos de políticas de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo de receta de componentes con una política de autorización

El siguiente ejemplo de receta de componentes incluye un objeto `accessControl` que define una política de autorización. Esta política autoriza al componente `com.example.HelloWorld` a publicar en el tema `test/topic`.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "java -jar {artifacts:path}/HelloWorld.jar"
      }
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
            - "test/topic"
Manifests:
  - Lifecycle:
      Run: |-
        java -jar {artifacts:path}/HelloWorld.jar
```

Example Ejemplo de actualización de la configuración de un componente con una política de autorización

El siguiente ejemplo de actualización de configuración en una implementación especifica la configuración de un componente con un objeto `accessControl` que define una política de autorización. Esta política autoriza al componente `com.example.HelloWorld` a publicar en el tema `test/topic`.

Console

Configuración de combinación

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
```

```
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

El siguiente comando crea una implementación a un dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

El archivo `hello-world-deployment.json` contiene el siguiente documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":
{\"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
 publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
\"resources\":[\"test/topic\"]}}}}"
      }
    }
  }
}
```

Greengrass CLI

El siguiente comando de la CLI de Greengrass crea una implementación local en un dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

El archivo `hello-world-configuration.json` contiene el siguiente documento JSON.

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  }
}
```

Suscripción a los flujos de eventos de IPC

Puede utilizar las operaciones de IPC para suscribirse a los flujos de eventos en un dispositivo principal de Greengrass. Para utilizar una operación de suscripción, defina un controlador de suscripciones y cree una solicitud al servicio de IPC. A continuación, el cliente de IPC ejecuta las funciones del controlador de suscripciones cada vez que el dispositivo principal transmite un mensaje de evento a su componente.

Puede cerrar una suscripción para dejar de procesar los mensajes de eventos. Para ello, llame a `closeStream()` (Java), `close()` (Python) o `Close()` (C++) en el objeto de operación de suscripción que utilizó para abrir la suscripción.

El servicio AWS IoT Greengrass Core IPC admite las siguientes operaciones de suscripción:

- SubscribeToTopic
- SubscribeToIoTCore
- SubscribeToComponentUpdates
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates

Temas

- Definición de controladores de suscripción
- Ejemplos de controladores de suscripciones

Definición de controladores de suscripción

Para definir un controlador de suscripciones, defina las funciones de devolución de llamada que gestionen los mensajes de eventos, los errores y el cierre de flujos. Si utiliza el cliente de IPC V1, debe definir estas funciones en una clase. Si usa el cliente IPC V2, que está disponible en versiones posteriores de Java y Python SDKs, puede definir estas funciones sin crear una clase de controlador de suscripciones.

Java

Si utiliza el cliente IPC V1, debe implementar la interfaz genérica. `software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<`*`StreamEventType`*`>` *`StreamEventType`*es el tipo de mensaje de evento para la operación de suscripción. Defina las siguientes funciones para gestionar los mensajes de eventos, los errores y el cierre de flujos.

Si usa el cliente de IPC V2, puede definir estas funciones fuera de una clase de controlador de suscripciones o usar expresiones lambda.

```
void onStreamEvent(StreamEventType event)
```

La llamada de retorno a la que llama el cliente de IPC cuando recibe un mensaje de evento, como un mensaje MQTT o una notificación de actualización de un componente.

```
boolean onStreamError(Throwable error)
```

La devolución de la llamada a la que llama el cliente de IPC cuando se produce un error de flujo.

Devuelve true para cerrar el flujo de suscripción como resultado del error, o devuelve false para mantener el flujo abierto.

```
void onStreamClosed()
```

La devolución de llamada a la que llama el cliente de IPC cuando se cierra el flujo.

Python

Si utiliza el cliente de IPC V1, debe ampliar la clase de controlador de respuesta de flujo que corresponde a la operación de suscripción. SDK para dispositivos con AWS IoT Incluye una clase de controlador de suscripciones para cada operación de suscripción. *StreamEventType*es el tipo de mensaje de evento para la operación de suscripción. Defina las siguientes funciones para gestionar los mensajes de eventos, los errores y el cierre de flujos.

Si usa el cliente de IPC V2, puede definir estas funciones fuera de una clase de controlador de suscripciones o usar expresiones lambda.

```
def on_stream_event(self, event: StreamEventType) -> None
```

La llamada de retorno a la que llama el cliente de IPC cuando recibe un mensaje de evento, como un mensaje MQTT o una notificación de actualización de un componente.

```
def on_stream_error(self, error: Exception) -> bool
```

La devolución de la llamada a la que llama el cliente de IPC cuando se produce un error de flujo.

Devuelve true para cerrar el flujo de suscripción como resultado del error, o devuelve false para mantener el flujo abierto.

```
def on_stream_closed(self) -> None
```

La devolución de llamada a la que llama el cliente de IPC cuando se cierra el flujo.

C++ (IPC client V1)

Implemente una clase que se derive de la clase del controlador de respuesta de flujo que corresponde a la operación de suscripción. SDK para dispositivos con AWS IoT Incluye una clase base de controlador de suscripciones para cada operación de suscripción. *StreamEventType*es el tipo de mensaje de evento para la operación de suscripción. Defina las siguientes funciones para gestionar los mensajes de eventos, los errores y el cierre de flujos.

`void OnStreamEvent(StreamEventType *event)`

La llamada de retorno a la que llama el cliente de IPC cuando recibe un mensaje de evento, como un mensaje MQTT o una notificación de actualización de un componente.

`bool OnStreamError(OperationError *error)`

La devolución de la llamada a la que llama el cliente de IPC cuando se produce un error de flujo.

Devuelve true para cerrar el flujo de suscripción como resultado del error, o devuelve false para mantener el flujo abierto.

`void OnStreamClosed()`

La devolución de llamada a la que llama el cliente de IPC cuando se cierra el flujo.

JavaScript

Implemente una clase que se derive de la clase del controlador de respuesta de flujo que corresponde a la operación de suscripción. SDK para dispositivos con AWS IoT Incluye una clase base de controlador de suscripciones para cada operación de suscripción. *StreamEventType*es el tipo de mensaje de evento para la operación de suscripción. Defina las siguientes funciones para gestionar los mensajes de eventos, los errores y el cierre de flujos.

`on(event: 'ended', listener: StreamingOperationEndedListener)`

La devolución de llamada a la que llama el cliente de IPC cuando se cierra el flujo.

`on(event: 'streamError', listener: StreamingRpcErrorListener)`

La devolución de la llamada a la que llama el cliente de IPC cuando se produce un error de flujo.

Devuelve true para cerrar el flujo de suscripción como resultado del error, o devuelve false para mantener el flujo abierto.

`on(event: 'message', listener: (message: InboundMessageType) => void)`

La llamada de retorno a la que llama el cliente de IPC cuando recibe un mensaje de evento, como un mensaje MQTT o una notificación de actualización de un componente.

Ejemplos de controladores de suscripciones

En el siguiente ejemplo, se muestra cómo utilizar la operación SubscribeToTopic y un controlador de suscripciones para suscribirse a la mensajería de publicación y suscripción local.

Java (IPC client V2)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);
```

```
            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
+ topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
```

```
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')
```

```
        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++ (IPC client V1)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>
```

```
using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
```

```
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
```

```
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
```

```
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
```

```
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Rust

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
use gg_sdk::{Sdk, SubscribeToTopicPayload};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";

    let callback = |topic: &str, payload: SubscribeToTopicPayload| match payload
    {
        SubscribeToTopicPayload::Binary(message) => {
            let message = String::from_utf8_lossy(message);
            println!("Received new message on topic {topic}: {message}");
        }
        SubscribeToTopicPayload::Json(_) => {
            println!("Received new message on topic {topic}: (JSON message)");
        }
    };

    let _sub = sdk
        .subscribe_to_topic(topic, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
```

```
        thread::sleep(Duration::from_secs(10));
    }
}
```

C

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
#include <assert.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <gg/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    if (gg_obj_type(payload) == GG_TYPE_BUF) {
        GgBuffer message = gg_obj_into_buf(payload);
        printf(
            "Received new message on topic %.*s: %.*s\n",
            (int) topic.len,
            topic.data,
            (int) message.len,
            message.data
        );
    } else {
        assert(gg_obj_type(payload) == GG_TYPE_MAP);
        printf(
            "Received new message on topic %.*s: (JSON message)\n",
            (int) topic.len,
            topic.data
        );
    }
}

int main(void) {
```

```
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_topic(
        topic, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the stream.
    ggipc_close_subscription(handle);
}
```

C++ (Component SDK)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
#include <gg/ipc/client.hpp>
#include <gg/object.hpp>
#include <unistd.h>
```

```
#include <cassert>
#include <iostream>

class ResponseHandler : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        if (payload.index() == GG_TYPE_BUF) {
            std::cout << "Received new message on topic " << topic << ": "
                      << get<gg::Buffer>(payload) << "\n";
        } else {
            assert(payload.index() == GG_TYPE_MAP);
            std::cout << "Received new message on topic " << topic
                      << ": (JSON message)\n";
        }
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";

    static ResponseHandler handler;
    error = client.subscribe_to_topic(topic, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
```

```
    }
}
```

Prácticas recomendadas de IPC

Las prácticas recomendadas para utilizar el IPC en los componentes personalizados difieren entre el cliente de IPC V1 y el cliente de IPC V2. Siga las prácticas recomendadas para la versión del cliente de IPC que se utilice.

IPC client V2

El cliente de IPC V2 ejecuta las funciones de devolución de llamadas en un subproceso independiente, por lo que, en comparación con el cliente de IPC V1, hay menos pautas que seguir cuando utiliza IPC y funciones del controlador de suscripciones.

- Reutilización de un cliente de IPC

  Después de crear un cliente de IPC, manténgalo abierto y reutilícelo para todas las operaciones de IPC. La creación de varios clientes utiliza recursos adicionales y puede provocar pérdidas de recursos.
- Tratamiento de excepciones

  El cliente de IPC V2 registra las excepciones no detectadas en las funciones del controlador de suscripciones. Debe detectar las excepciones en las funciones de su controlador para gestionar los errores que se producen en su código.

IPC client V1

El cliente de IPC V1 utiliza un único subproceso que se comunica con el servidor de IPC y llama a los controladores de suscripciones. Debe tener en cuenta este comportamiento sincrónico al escribir las funciones del controlador de suscripciones.

- Reutilización de un cliente de IPC

  Después de crear un cliente de IPC, manténgalo abierto y reutilícelo para todas las operaciones de IPC. La creación de varios clientes utiliza recursos adicionales y puede provocar pérdidas de recursos.
- Ejecución del código de bloqueo de forma asincrónica

El cliente del PC V1 no puede enviar nuevas solicitudes ni procesar nuevos mensajes de eventos mientras el subproceso está bloqueado. Debe ejecutar el código de bloqueo en un subproceso independiente que ejecute desde la función de controlador. El código de bloqueo incluye `sleep` llamadas, bucles que se ejecutan de forma continua y I/O solicitudes sincrónicas que tardan en completarse.

- Envío de nuevas solicitudes de IPC de forma asincrónica

  El cliente de IPC V1 no puede enviar una nueva solicitud desde las funciones del controlador de suscripciones, ya que la solicitud bloquea la función del controlador si se espera una respuesta. Debe enviar las solicitudes de IPC en un subproceso independiente que ejecute desde la función de controlador.

- Tratamiento de excepciones

  El cliente de IPC V1 no controla las excepciones no detectadas en las funciones del controlador de suscripciones. Si su función de controlador genera una excepción, la suscripción se cierra y la excepción no aparece en los registros de sus componentes. Debe detectar las excepciones en las funciones de su controlador para mantener la suscripción abierta y registrar los errores que se produzcan en el código.

Publicar/suscribir mensajes locales

La mensajería de publicación y suscripción (pubsub) le permite enviar y recibir mensajes sobre temas. Los componentes pueden publicar mensajes en los temas para enviar mensajes a otros componentes. A continuación, los componentes que están suscritos a ese tema pueden actuar en función de los mensajes que reciben.

> **Note**
>
> No puede utilizar este servicio de publish/subscribe IPC para publicar o suscribirse a AWS IoT Core MQTT. Para obtener más información sobre cómo intercambiar mensajes con AWS IoT Core MQTT, consulte. Publicar/suscribir mensajes MQTT AWS IoT Core

Temas

- Versiones mínimas de SDK
- Autorización

- PublishToTopic
- SubscribeToTopic
- Ejemplos

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para publicar y suscribirse a los mensajes de temas locales y desde ellos.

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Java v2	Versión 1.2.10
SDK para dispositivos con AWS IoT para Python v2	Versión 1.5.3
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0
SDK de AWS IoT Device para JavaScript v2	Versión 1.12.0

Autorización

Para utilizar la publish/subscribe mensajería local en un componente personalizado, debe definir políticas de autorización que permitan a su componente enviar y recibir mensajes sobre los temas. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización publish/subscribe de mensajería tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.ipc.pubsub`

Operación	Description (Descripción)	Recursos
`aws.greengrass#PublishToTopic`	Permite que un componente publique mensajes en los temas que especifique.	Una cadena de tema, como `test/topic` . Use un * para hacer coincidir cualquier combinación de caracteres de un tema. Esta cadena de tema no admite los caracteres comodín (# y +) de los temas MQTT.
`aws.greengrass#SubscribeToTopic`	Permite que un componente se suscriba a los mensajes de los temas que especifique.	Una cadena de tema, como `test/topic` . Use un * para hacer coincidir cualquier combinación de caracteres de un tema. En el núcleo de Greengrass versión 2.6.0 y versiones posteriores, puede suscribirse a temas que contengan comodines de temas MQTT (# y +). Esta cadena de tema admite los comodines de los temas MQTT como caracteres literales. Por ejemplo, si la política de autorización de un componente permite el acceso a `test/topic/#`, el componente se puede suscribir a `test/topic/#` , pero no se puede suscribir a `test/topic/filter` .
*	Permite que un componente publique y se suscriba a los	Una cadena de tema, como `test/topic` . Use un *

Operación	Description (Descripción)	Recursos
	mensajes de los temas que especifique.	para hacer coincidir cualquier combinación de caracteres de un tema. En el núcleo de Greengrass versión 2.6.0 y versiones posteriores, puede suscribirse a temas que contengan comodines de temas MQTT (# y +). Esta cadena de tema admite los comodines de los temas MQTT como caracteres literales. Por ejemplo, si la política de autorización de un componente permite el acceso a `test/topic/#`, el componente se puede suscribir a `test/topic/#` , pero no se puede suscribir a `test/topic/filter` .

Ejemplos de políticas de autorización

Puede consultar el siguiente ejemplo de política de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo de política de autorización

El siguiente ejemplo de política de autorización permite a un componente publicar y suscribirse a todos los temas.

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyLocalPubSubComponent:pubsub:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
```

```
          "aws.greengrass#PublishToTopic",
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

PublishToTopic

Publique un mensaje en un tema

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`topic`

El tema en el que se va a publicar el mensaje.

`publishMessage` (Python: `publish_message`)

El mensaje a publicar. Este objeto, `PublishMessage`, contiene la siguiente información. Debe especificar uno entre `jsonMessage` y `binaryMessage`.

`jsonMessage` (Python: `json_message`)

(Opcional) Un mensaje JSON. Este objeto, `JsonMessage`, contiene la siguiente información:

`message`

El mensaje JSON como un objeto.

`context`

El contexto del mensaje, como el tema en el que se publicó el mensaje.

Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. En la siguiente tabla, se enumeran las versiones mínimas de SDK para dispositivos con AWS IoT que debe utilizar para acceder al contexto del mensaje.

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.18.4
SDK de AWS IoT Device para JavaScript v2	Versión 1.12.0

> **Note**
>
> El software AWS IoT Greengrass Core utiliza los mismos objetos de mensaje en las `SubscribeToTopic` operaciones `PublishToTopic` y. El software AWS IoT Greengrass Core establece este objeto de contexto en los mensajes cuando te suscribes e ignora este objeto de contexto en los mensajes que publicas.

Este objeto, `MessageContext`, contiene la siguiente información:

`topic`

El tema donde se publicó el mensaje.

`binaryMessage` (Python: `binary_message`)

(Opcional) Un mensaje binario. Este objeto, `BinaryMessage`, contiene la siguiente información:

`message`

El mensaje binario es un blob.

`context`

El contexto del mensaje, como el tema en el que se publicó el mensaje.

Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. En la siguiente tabla, se enumeran las versiones mínimas de SDK para dispositivos con AWS IoT que debe utilizar para acceder al contexto del mensaje.

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.18.4
SDK de AWS IoT Device para JavaScript v2	Versión 1.12.0

> **Note**
>
> El software AWS IoT Greengrass Core utiliza los mismos objetos de mensaje en las `SubscribeToTopic` operaciones `PublishToTopic` y. El software AWS IoT Greengrass Core establece este objeto de contexto en los mensajes cuando te suscribes e ignora este objeto de contexto en los mensajes que publicas.

Este objeto, `MessageContext`, contiene la siguiente información:

`topic`

El tema donde se publicó el mensaje.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V2)

Example Ejemplo: publicar un mensaje binario

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
 message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
```

```
            GreengrassCoreIPCClientV2 ipcClient, String topic, String message)
 throws InterruptedException {
        BinaryMessage binaryMessage =
                new
 BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
 PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
                new
 PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}
```

Python (IPC client V2)

Example Ejemplo: publicar un mensaje binario

```
import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)


def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def publish_binary_message_to_topic(ipc_client, topic, message):
```

```
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
 publish_message=publish_message)


if __name__ == '__main__':
    main()
```

C++ (IPC client V1)

Example Ejemplo: publicar un mensaje binario

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
```

```
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    String message("Hello, World!");
    int timeout = 10;

    PublishToTopicRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    BinaryMessage binaryMessage;
    binaryMessage.SetMessage(messageData);
    PublishMessage publishMessage;
    publishMessage.SetBinaryMessage(binaryMessage);
    request.SetTopic(topic);
    request.SetPublishMessage(publishMessage);

    auto operation = ipcClient.NewPublishToTopic();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }
    return 0;
```

```
}
```

JavaScript

Example Ejemplo: publicar un mensaje binario

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";

class PublishToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;
    private readonly messageString : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.messageString = "<define_your_message_string>";
        this.publishToTopic().then(r => console.log("Started workflow"));
    }

    private async publishToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const binaryMessage : BinaryMessage = {
                message: this.messageString
            }

            const publishMessage : PublishMessage = {
                binaryMessage: binaryMessage
            }

            const request : PublishToTopicRequest = {
                topic: this.topic,
                publishMessage: publishMessage
            }

            this.ipcClient.publishToTopic(request).finally(() =>
 console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))

        } catch (e) {
```

```
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();
```

Rust

Example Ejemplo: publicar un mensaje binario

```
use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";

    sdk.publish_to_topic_binary(topic, message)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}
```

C

Example Ejemplo: publicar un mensaje binario

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");

    err = ggipc_publish_to_topic_binary(topic, message);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}
```

C++ (Component SDK)

Example Ejemplo: publicar un mensaje binario

```
#include <gg/ipc/client.hpp>
```

```
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";

    error = client.publish_to_topic(topic, message);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}
```

SubscribeToTopic

Suscripción a los mensajes sobre un tema.

Esta es una operación de suscripción en la que se suscribe a un flujo de mensajes de eventos. Para usar esta operación, defina un identificador de respuesta de flujo con funciones que gestionen los mensajes de eventos, los errores y el cierre del flujo. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Tipo de mensaje del evento: `SubscriptionResponseMessage`

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`topic`

El tema al que se suscribe.

> **Note**
>
> En el núcleo de Greengrass versión 2.6.0 y versiones posteriores, este tema admite los comodines (# y +) de los temas MQTT.

`receiveMode` (Python: `receive_mode`)

(Opcional) El comportamiento que especifica si el componente recibe mensajes de sí mismo. Puede cambiar este comportamiento para permitir que un componente actúe sobre sus propios mensajes. El comportamiento predeterminado depende de si el tema contiene un comodín MQTT. Puede elegir entre las siguientes opciones:

- `RECEIVE_ALL_MESSAGES`: reciba todos los mensajes que coincidan con el tema, incluidos los mensajes del componente al que se suscribe.

  Este modo es la opción por defecto cuando se suscribe a un tema que no contiene un comodín MQTT.

- `RECEIVE_MESSAGES_FROM_OTHERS`: reciba todos los mensajes que coincidan con el tema, excepto los del componente al que se suscribe.

  Este modo es la opción por defecto cuando se suscribe a un tema que contiene un comodín MQTT.

Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. En la siguiente tabla se enumeran las versiones mínimas del SDK para dispositivos con AWS IoT que debe utilizar para configurar el modo de recepción.

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.18.4

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	

Respuesta

Esta respuesta de operación contiene la siguiente información:

`messages`

El flujo de mensajes. Este objeto, `SubscriptionResponseMessage`, contiene la siguiente información. Cada mensaje contiene `jsonMessage` o `binaryMessage`.

`jsonMessage` (Python: `json_message`)

(Opcional) Un mensaje JSON. Este objeto, `JsonMessage`, contiene la siguiente información:

`message`

El mensaje JSON como un objeto.

`context`

El contexto del mensaje, como el tema en el que se publicó el mensaje.

Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. En la siguiente tabla, se enumeran las versiones mínimas de SDK para dispositivos con AWS IoT que debe utilizar para acceder al contexto del mensaje.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.18.4	

SDK	Versión mínima	
SDK de AWS IoT Device para JavaScript v2	Versión 1.12.0	

> **Note**
>
> El software AWS IoT Greengrass Core utiliza los mismos objetos de mensaje en las `SubscribeToTopic` operaciones `PublishToTopic` y. El software AWS IoT Greengrass Core establece este objeto de contexto en los mensajes cuando te suscribes e ignora este objeto de contexto en los mensajes que publicas.

Este objeto, `MessageContext`, contiene la siguiente información:

`topic`

El tema donde se publicó el mensaje.

`binaryMessage` (Python: `binary_message`)

(Opcional) Un mensaje binario. Este objeto, `BinaryMessage`, contiene la siguiente información:

`message`

El mensaje binario es un blob.

`context`

El contexto del mensaje, como el tema en el que se publicó el mensaje.

Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. En la siguiente tabla, se enumeran las versiones mínimas de SDK para dispositivos con AWS IoT que debe utilizar para acceder al contexto del mensaje.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3	

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.18.4
SDK de AWS IoT Device para JavaScript v2	Versión 1.12.0

> **Note**
>
> El software AWS IoT Greengrass Core utiliza los mismos objetos de mensaje en las `SubscribeToTopic` operaciones `PublishToTopic` y. El software AWS IoT Greengrass Core establece este objeto de contexto en los mensajes cuando te suscribes e ignora este objeto de contexto en los mensajes que publicas.

Este objeto, `MessageContext`, contiene la siguiente información:

`topic`

El tema donde se publicó el mensaje.

`topicName` (Python: `topic_name`)

El tema en el que se publicó el mensaje.

> **Note**
>
> En la actualidad, esta propiedad no se utiliza. En el núcleo de Greengrass versión 2.6.0 y versiones posteriores, puede obtener el valor `(jsonMessage|binaryMessage).context.topic` de un `SubscriptionResponseMessage` para obtener el tema en el que se publicó el mensaje.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V2)

Example Ejemplo: suscríbase a los mensajes locales publish/subscribe

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
```

```
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
+ topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
```

```
}
```

Python (IPC client V2)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
```

```
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++ (IPC client V1)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}
```

```
    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
```

```
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
```

```
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }
```

```
            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
```

```
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Rust

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
use gg_sdk::{Sdk, SubscribeToTopicPayload};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";

    let callback = |topic: &str, payload: SubscribeToTopicPayload| match payload
    {
        SubscribeToTopicPayload::Binary(message) => {
            let message = String::from_utf8_lossy(message);
            println!("Received new message on topic {topic}: {message}");
        }
        SubscribeToTopicPayload::Json(_) => {
            println!("Received new message on topic {topic}: (JSON message)");
        }
    };

    let _sub = sdk
        .subscribe_to_topic(topic, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

C

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
#include <assert.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <gg/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    if (gg_obj_type(payload) == GG_TYPE_BUF) {
        GgBuffer message = gg_obj_into_buf(payload);
        printf(
            "Received new message on topic %.*s: %.*s\n",
            (int) topic.len,
            topic.data,
            (int) message.len,
            message.data
        );
    } else {
        assert(gg_obj_type(payload) == GG_TYPE_MAP);
        printf(
            "Received new message on topic %.*s: (JSON message)\n",
            (int) topic.len,
            topic.data
        );
    }
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
```

```
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_topic(
        topic, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the stream.
    ggipc_close_subscription(handle);
}
```

C++ (Component SDK)

Example Ejemplo: suscríbase a los publish/subscribe mensajes locales

```
#include <gg/ipc/client.hpp>
#include <gg/object.hpp>
#include <unistd.h>
#include <cassert>
#include <iostream>

class ResponseHandler : public gg::ipc::LocalTopicCallback {
```

```
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        if (payload.index() == GG_TYPE_BUF) {
            std::cout << "Received new message on topic " << topic << ": "
                      << get<gg::Buffer>(payload) << "\n";
        } else {
            assert(payload.index() == GG_TYPE_MAP);
            std::cout << "Received new message on topic " << topic
                      << ": (JSON message)\n";
        }
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";

    static ResponseHandler handler;
    error = client.subscribe_to_topic(topic, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}
```

Ejemplos

Utilice los siguientes ejemplos para aprender a utilizar el servicio publish/subscribe IPC en sus componentes.

Ejemplo de publish/subscribe editor (Java, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherJava:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "java -jar {artifacts:path}/PubSubPublisher.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      Run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar
```

El siguiente ejemplo de aplicación Java demuestra cómo utilizar el servicio IPC de publicación/suscripción para publicar mensajes en otros componentes.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
```

```
public class PubSubPublisher {

    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                        .publishToTopic(publishRequest,
 Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
                    System.err.println("Timeout occurred while publishing to topic: " +
 topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
 topic: " + topic);
                    }
                    throw e;
                }
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            System.out.println("Publisher interrupted.");
        } catch (Exception e) {
```

```
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Ejemplo de publish/subscribe suscriptor (Java, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
      }
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      Run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

El siguiente ejemplo de aplicación Java muestra cómo utilizar el servicio IPC de publicación/suscripción para suscribirse a los mensajes de otros componentes.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                    .subscribeToTopic(subscribeRequest, Optional.of(new
 SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
 operationResponseHandler.getResponse();

            try {
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
 " + topic);
                } else {
                    System.err.println("Execution exception while subscribing to topic:
 " + topic);
                }
                throw e;
            }

            // Keep the main thread alive, or the process will exit.
```

```
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static class SubscribeResponseHandler implements
 StreamResponseHandler<SubscriptionResponseMessage> {

        @Override
        public void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
            try {
                String message = new
 String(subscriptionResponseMessage.getBinaryMessage()
                        .getMessage(), StandardCharsets.UTF_8);
                System.out.println("Received new message: " + message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false; // Return true to close stream, false to keep stream open.
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to topic stream closed.");
        }
    }
}
```

Ejemplo de publish/subscribe editor (Python, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "Run": "python3 -u {artifacts:path}/pubsub_publisher.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
```

```
        "install": "py -3 -m pip install --user awsiotsdk",
        "Run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      Run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      Run: py -3 -u {artifacts:path}/pubsub_publisher.py
```

El siguiente ejemplo de aplicación de Python demuestra cómo utilizar el servicio IPC de publicación/suscripción para publicar mensajes en otros componentes.

```
import concurrent.futures
import sys
```

```
import time
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)


topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
 file=sys.stderr)
        except UnauthorizedError as e:
            print('Unauthorized error while publishing to topic: ' + topic,
 file=sys.stderr)
            raise e
        except Exception as e:
            print('Exception while publishing to topic: ' + topic, file=sys.stderr)
            raise e
        time.sleep(5)
```

```
except InterruptedError:
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Ejemplo de publish/subscribe suscriptor (Python, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "Run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
```

```
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "Run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      Run: python3 -u {artifacts:path}/pubsub_subscriber.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      Run: py -3 -u {artifacts:path}/pubsub_subscriber.py
```

El siguiente ejemplo de aplicación de Python demuestra cómo utilizar el servicio IPC de publicación/suscripción para suscribirse a los mensajes de otros componentes.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10


class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()
        return False  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        print('Subscribe to topic stream closed.')


try:
    ipc_client = awsiot.greengrasscoreipc.connect()
```

```
    request = SubscribeToTopicRequest()
    request.topic = topic
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Ejemplo de publish/subscribe editor (C++, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
```

```
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
```

```
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio IPC de publicación/suscripción para publicar mensajes en otros componentes.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
```

```
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }
```

```
        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

Ejemplo de publish/subscribe suscriptor (C++, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
```

```
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
```

```
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio IPC de publicación/suscripción para suscribirse a los mensajes de otros componentes.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                std::cout << "Received new message: " << messageString << std::endl;
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
```

```
                    std::cout << "Received new message: " << messageString <<
 std::endl;
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to topic stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
```

```
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
```

```
    }

    operation->Close();
    return 0;
}
```

Ejemplo de publish/subscribe editor (Rust)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherRust",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherRust:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/publish_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherRust/1.0.0/publish_to_topic",
          "Permission": {
            "Execute": "OWNER"
```

```
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación Rust demuestra cómo utilizar el servicio IPC de publicación/suscripción para publicar mensajes en otros componentes.

```
use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";

    sdk.publish_to_topic_binary(topic, message)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}
```

Ejemplo de publish/subscribe suscriptor (Rust)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberRust",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberRust:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToTopic"],
            "resources": ["*"]
```

```
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/subscribe_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberRust/1.0.0/subscribe_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación Rust demuestra cómo utilizar el servicio IPC de publicación/suscripción para suscribirse a mensajes de otros componentes.

```
use gg_sdk::{Sdk, SubscribeToTopicPayload};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";

    let callback = |topic: &str, payload: SubscribeToTopicPayload| match payload
    {
        SubscribeToTopicPayload::Binary(message) => {
            let message = String::from_utf8_lossy(message);
```

```
            println!("Received new message on topic {topic}: {message}");
        }
        SubscribeToTopicPayload::Json(_) => {
            println!("Received new message on topic {topic}: (JSON message)");
        }
    };

    let _sub = sdk
        .subscribe_to_topic(topic, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

Ejemplo de publish/subscribe editor (C)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherC",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherC:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
```

```
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_publish_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherC/1.0.0/sample_publish_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación en C muestra cómo utilizar el servicio IPC de publicación/suscripción para publicar mensajes en otros componentes.

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");

    err = ggipc_publish_to_topic_binary(topic, message);
    if (err != GG_ERR_OK) {
```

```
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}
```

Ejemplo de publish/subscribe suscriptor (C)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberC",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberC:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
```

```
        "run": "{artifacts:path}/sample_subscribe_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberC/1.0.0/sample_subscribe_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación en C muestra cómo utilizar el servicio IPC de publicación/suscripción para suscribirse a los mensajes de otros componentes.

```
#include <assert.h>
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <gg/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgObject payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    if (gg_obj_type(payload) == GG_TYPE_BUF) {
        GgBuffer message = gg_obj_into_buf(payload);
        printf(
            "Received new message on topic %.*s: %.*s\n",
            (int) topic.len,
            topic.data,
            (int) message.len,
            message.data
        );
```

```
    } else {
        assert(gg_obj_type(payload) == GG_TYPE_MAP);
        printf(
            "Received new message on topic %.*s: (JSON message)\n",
            (int) topic.len,
            topic.data
        );
    }
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_topic(
        topic, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
```

```
    // To stop subscribing, close the stream.
    ggipc_close_subscription(handle);
}
```

Ejemplo de publish/subscribe editor (C++, SDK de componentes)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_cpp_publish_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubPublisherCpp/1.0.0/sample_cpp_publish_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
```

```
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio IPC de publicación/suscripción para publicar mensajes en otros componentes.

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";

    error = client.publish_to_topic(topic, message);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}
```

Ejemplo de publish/subscribe suscriptor (C++, SDK de componentes)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
```

```
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToTopic"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_cpp_subscribe_to_topic"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/sample_cpp_subscribe_to_topic",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio IPC de publicación/suscripción para suscribirse a los mensajes de otros componentes.

```
#include <gg/ipc/client.hpp>
#include <gg/object.hpp>
#include <unistd.h>
#include <cassert>
#include <iostream>
```

```
class ResponseHandler : public gg::ipc::LocalTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Object payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        if (payload.index() == GG_TYPE_BUF) {
            std::cout << "Received new message on topic " << topic << ": "
                      << get<gg::Buffer>(payload) << "\n";
        } else {
            assert(payload.index() == GG_TYPE_MAP);
            std::cout << "Received new message on topic " << topic
                      << ": (JSON message)\n";
        }
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";

    static ResponseHandler handler;
    error = client.subscribe_to_topic(topic, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}
```

Publicar/suscribir mensajes MQTT AWS IoT Core

El servicio IPC de mensajería AWS IoT Core MQTT le permite enviar y recibir mensajes MQTT de ida y vuelta. AWS IoT Core Los componentes pueden publicar mensajes en los temas AWS IoT Core y suscribirse a ellos para actuar en función de los mensajes MQTT de otras fuentes. Para obtener más información sobre la AWS IoT Core implementación de MQTT, consulte MQTT en la AWS IoT Core Guía para desarrolladores.

> **Note**
>
> Este servicio IPC de mensajería MQTT le permite intercambiar mensajes con. AWS IoT Core Para obtener más información sobre cómo intercambiar mensajes entre componentes, consulte Publicar/suscribir mensajes locales.

Temas

- Versiones mínimas de SDK
- Autorización
- PublishToIoTCore
- SubscribeToIoTCore
- Ejemplos

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para publicar y recibir mensajes MQTT y suscribirse a ellos. AWS IoT Core

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.2.10	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.5.3	

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0	
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	

Autorización

Para utilizar la mensajería AWS IoT Core MQTT en un componente personalizado, debe definir políticas de autorización que permitan a su componente enviar y recibir mensajes sobre temas. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización para la mensajería AWS IoT Core MQTT tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.ipc.mqttproxy`

Operación	Description (Descripción)	Recursos
`aws.greengrass#PublishToIoTCore`	Permite que un componente publique mensajes AWS IoT Core en los temas de MQTT que especifique.	Una cadena de temas, por ejemplo, `test/topic` , o `*` para permitir el acceso a todos los temas. Puede utilizar los caracteres comodín (# y +) de los temas de MQTT para hacer coincidir varios recursos.
`aws.greengrass#SubscribeToIoTCore`	Permite que un componente se suscriba a los mensajes AWS IoT Core de los temas que especifique.	Una cadena de temas, por ejemplo, `test/topic` , o `*` para permitir el acceso a todos los temas. Puede utilizar los caracteres comodín (# y +) de los temas de MQTT

Operación	Description (Descripción)	Recursos
		para hacer coincidir varios recursos.
*	Permite que un componente publique y se suscriba a los mensajes de AWS IoT Core MQTT para los temas que especifique.	Una cadena de temas, por ejemplo, `test/topic` , o * para permitir el acceso a todos los temas. Puede utilizar los caracteres comodín (# y +) de los temas de MQTT para hacer coincidir varios recursos.

Comodín de MQTT en AWS IoT Core las políticas de autorización de MQTT

Puede utilizar caracteres comodín de MQTT en AWS IoT Core las políticas de autorización de IPC de MQTT. Los componentes pueden publicar y suscribirse a temas que coincidan con el filtro de temas que usted permita en una política de autorización. Por ejemplo, si la política de autorización de un componente concede acceso a `test/topic/#`, el componente puede suscribirse a `test/topic/#`, publicar y suscribirse a `test/topic/filter`.

Variables de receta en las políticas de autorización de MQTT AWS IoT Core

Si usa la versión 2.6.0 o posterior del núcleo de Greengrass, puede usar la variable de receta `{iot:thingName}` en las políticas de autorización. Esta característica le permite configurar una política de autorización única para un grupo de dispositivos principales, de forma que cada dispositivo principal solo pueda acceder a los temas que contengan su propio nombre. Por ejemplo, puede permitir que un componente acceda al siguiente recurso de tema.

```
devices/{iot:thingName}/messages
```

Para obtener más información, consulte Variables de receta y Uso de variables de receta en las actualizaciones de combinación.

Ejemplos de políticas de autorización

Puede consultar los siguientes ejemplos de políticas de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo de política de autorización con acceso ilimitado

El siguiente ejemplo de política de autorización permite a un componente publicar y suscribirse a todos los temas.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - "*"
```

Example Ejemplo de política de autorización con acceso limitado

El siguiente ejemplo de política de autorización permite a un componente publicar y suscribirse a dos temas denominados `factory/1/events` y `factory/1/actions`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1
 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/actions
        - factory/1/events
```

Example Ejemplo de política de autorización para un grupo de dispositivos principales

> **Important**
>
> En este ejemplo, se utiliza una característica que está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. El núcleo de Greengrass versión 2.6.0 suma compatibilidad con la mayoría de las variables de receta, por ejemplo: `{iot:thingName}`, en las configuraciones de componentes.

El siguiente ejemplo de política de autorización permite a un componente publicar y suscribirse a un tema que contenga el nombre del dispositivo principal que ejecuta el componente.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
```

```
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/devices/{iot:thingName}/controls
```

PublishToIoTCore

Publica un mensaje de MQTT AWS IoT Core sobre un tema.

Al publicar mensajes MQTT en AWS IoT Core, hay una cuota de 100 transacciones por segundo. Si supera esta cuota, los mensajes se ponen en cola para su procesamiento en el dispositivo de Greengrass. También hay una cuota de 512 KB de datos por segundo y una cuota de 20 000 publicaciones por segundo en toda la cuenta (2 000 en algunos casos). Regiones de AWS Para obtener más información sobre el límite del agente de mensajes MQTT en AWS IoT Core, consulte Límites de protocolo y cuotas del agente de mensajes de AWS IoT Core.

Si superas estas cuotas, el dispositivo Greengrass limita la publicación de mensajes a. AWS IoT Core Los mensajes se almacenan en un spooler de memoria. De forma predeterminada, la memoria asignada al spooler es de 2,5 Mb. Si el spooler se llena, se rechazan los mensajes nuevos. Puede aumentar el tamaño del spooler. Para obtener más información, consulte Configuración en la documentación del Núcleo de Greengrass. Para evitar llenar el espacio y tener que aumentar la memoria asignada, limite las solicitudes de publicación a no más de 100 solicitudes por segundo.

Si su aplicación necesita enviar mensajes a una velocidad mayor o mensajes más grandes, considere utilizar el Administrador de flujos para enviar mensajes a Kinesis Data Streams. El componente administrador de flujos está diseñado para transferir grandes volúmenes de datos a la Nube de AWS. Para obtener más información, consulte Administración de flujos de datos en los dispositivos principales de Greengrass.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`topicName` (Python: `topic_name`)

El tema en el que se va a publicar el mensaje.

`qos`

La QoS de MQTT que se va a utilizar. Esta enumeración, `QOS`, tiene los siguientes valores:

- `AT_MOST_ONCE`: QoS 0. El mensaje MQTT se entrega como máximo una vez.
- `AT_LEAST_ONCE`: QoS 1. El mensaje MQTT se entrega como máximo una vez.

`payload`

(Opcional) El mensaje se carga como un blob.

Las siguientes características están disponibles para la versión 2.10.0 y versiones posteriores del Núcleo de Greengrass cuando se utiliza MQTT 5. Estas características no se tienen en cuenta al utilizar MQTT 3.1.1. La siguiente tabla muestra la versión mínima del SDK del AWS IoT dispositivo que debe usar para acceder a estas funciones.

SDK	Versión mínima
AWS IoT Device SDK para Python v2	Versión 1.15.0
AWS IoT Device SDK para Java v2	Versión 1.13.0
AWS IoT Device SDK para C++ v2	Versión 1.24.0
SDK de AWS IoT Device para JavaScript v2	Versión 1.13.0

`payloadFormat`

(Opcional) El formato de la carga útil del mensaje. Si no establece el `payloadFormat`, se supone que el tipo es `BYTES`. Esta enumeración tiene los siguientes valores:

- `BYTES`: el contenido de la carga útil es un blob binario.
- `UTF8`— El contenido de la carga útil es una UTF8 cadena de caracteres.

`retain`

(Opcional) Indica si se debe configurar la opción de retención de MQTT en `true` durante la publicación.

`userProperties`

(Opcional) Una lista de objetos `UserProperty` específicos de la aplicación para enviar. El objeto `UserProperty` se define de la siguiente manera:

```
UserProperty:
```

```
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(Opcional) La cantidad de segundos antes de que el mensaje venza y el servidor lo elimine. Si no se establece este valor, el mensaje no vence.

`correlationData`

(Opcional) Información agregada a la solicitud que se puede usar para asociar una solicitud a una respuesta.

`responseTopic`

(Opcional) El tema que debe usarse para el mensaje de respuesta.

`contentType`

(Opcional) Un identificador específico de la aplicación del tipo de contenido del mensaje.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V2)

Example Ejemplo: publicar un mensaje

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {
```

```
    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                    .withTopicName(topic)
                    .withPayload(message.getBytes(StandardCharsets.UTF_8))
                    .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example Ejemplo: publicar un mensaje

> **Note**
>
> En este ejemplo se supone que está utilizando la versión 1.5.4 o posterior de SDK para dispositivos con AWS IoT para Python v2.

```
import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()
```

Java (IPC client V1)

Example Ejemplo: publicar un mensaje

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            PublishToIoTCoreResponseHandler responseHandler =
```

```
                    PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
 message, qos);
            CompletableFuture<PublishToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully published to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while publishing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToIoTCoreResponseHandler
 publishBinaryMessageToTopic(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
 PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

 publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCClient.publishToIoTCore(publishToIoTCoreRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Ejemplo: publicar un mensaje

> **Note**
>
> En este ejemplo se supone que está utilizando la versión 1.5.4 o posterior de SDK para dispositivos con AWS IoT para Python v2.

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE

request = PublishToIoTCoreRequest()
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++ (IPC client V1)

Example Ejemplo: publicar un mensaje

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>
```

```
using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();
```

```
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
}
```

JavaScript

Example Ejemplo: publicar un mensaje

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
```

```
    }

    private async publishToIoTCore() {
        try {
            const request: PublishToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```

Rust

Example Ejemplo: publicar un mensaje

```
use gg_sdk::{Qos, Sdk};
```

```
fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;

    sdk.publish_to_iot_core(topic, message, qos)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}
```

C

Example Ejemplo: publicar un mensaje

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    err = ggipc_publish_to_iot_core(topic, message, qos);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
```

```
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}
```

C++ (Component SDK)

Example Ejemplo: publicar un mensaje

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    error = client.publish_to_iot_core(topic, message, qos);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}
```

SubscribeToIoTCore

Suscríbase a los mensajes de MQTT AWS IoT Core de un tema o filtro de temas. El software AWS IoT Greengrass Core elimina las suscripciones cuando el componente llega al final de su ciclo de vida.

Esta es una operación de suscripción en la que se suscribe a un flujo de mensajes de eventos. Para usar esta operación, defina un identificador de respuesta de flujo con funciones que gestionen los mensajes de eventos, los errores y el cierre del flujo. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Tipo de mensaje del evento: `IoTCoreMessage`

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`topicName` (Python: `topic_name`)

El tema al que se suscribe. Puede utilizar los comodines (# y +) de los temas de MQTT para suscribirse a varios temas.

`qos`

La QoS de MQTT que se va a utilizar. Esta enumeración, `QOS`, tiene los siguientes valores:

- `AT_MOST_ONCE`: QoS 0. El mensaje MQTT se entrega como máximo una vez.
- `AT_LEAST_ONCE`: QoS 1. El mensaje MQTT se entrega como máximo una vez.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`messages`

El flujo de mensajes MQTT. Este objeto, `IoTCoreMessage`, contiene la siguiente información:

`message`

El mensaje MQTT. Este objeto, `MQTTMessage`, contiene la siguiente información:

`topicName` (Python: `topic_name`)

El tema en el que se publicó el mensaje.

`payload`

(Opcional) El mensaje se carga como un blob.

Las siguientes características están disponibles para la versión 2.10.0 y versiones posteriores del Núcleo de Greengrass cuando se utiliza MQTT 5. Estas características no se tienen en cuenta al utilizar MQTT 3.1.1. En la siguiente tabla se muestra la versión mínima del SDK del AWS IoT dispositivo que debes usar para acceder a estas funciones.

SDK	Versión mínima
AWS IoT Device SDK para Python v2	Versión 1.15.0
AWS IoT Device SDK para Java v2	Versión 1.13.0
AWS IoT Device SDK para C++ v2	Versión 1.24.0
SDK de AWS IoT Device para JavaScript v2	Versión 1.13.0

`payloadFormat`

(Opcional) El formato de la carga útil del mensaje. Si no establece el `payloadFormat`, se supone que el tipo es `BYTES`. Esta enumeración tiene los siguientes valores:

- `BYTES`: el contenido de la carga útil es un blob binario.
- `UTF8`— El contenido de la carga útil es una UTF8 cadena de caracteres.

`retain`

(Opcional) Indica si se debe configurar la opción de retención de MQTT en `true` durante la publicación.

`userProperties`

(Opcional) Una lista de objetos `UserProperty` específicos de la aplicación para enviar. El objeto `UserProperty` se define de la siguiente manera:

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(Opcional) La cantidad de segundos antes de que el mensaje venza y el servidor lo elimine. Si no se establece este valor, el mensaje no vence.

`correlationData`

(Opcional) Información agregada a la solicitud que se puede usar para asociar una solicitud a una respuesta.

`responseTopic`

(Opcional) El tema que debe usarse para el mensaje de respuesta.

`contentType`

(Opcional) Un identificador específico de la aplicación del tipo de contenido del mensaje.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V2)

Example Ejemplo: suscribirse a los mensajes

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.function.Consumer;
import java.util.function.Function;


public class SubscribeToIoTCore {

    public static void main(String[] args) {
```

```
        String topic = args[0];
        QOS qos = QOS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = ioTCoreMessage ->
                System.out.printf("Received new message on topic %s: %s%n",
                        ioTCoreMessage.getMessage().getTopicName(),
                        new String(ioTCoreMessage.getMessage().getPayload(),
StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
                Optional.of(e -> {
                    System.err.println("Received a stream error.");
                    e.printStackTrace();
                    return false;
                });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
                System.out.println("Subscribe to IoT Core stream closed."));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                    .withTopicName(topic)
                    .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
SubscribeToIoTCoreResponseHandler>
                    streamingResponse = ipcClientV2.subscribeToIoTCore(request,
onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }

            // To stop subscribing, close the stream.
            streamingResponse.getHandler().closeStream();
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred.");
```

```
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example Ejemplo: suscribirse a los mensajes

> **Note**
>
> En este ejemplo se supone que está utilizando la versión 1.5.4 o posterior de SDK para dispositivos con AWS IoT para Python v2.

```
import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'

def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}:  {message}')
    except:
        traceback.print_exc()

def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():
    pass

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
```

```
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()
```

Java (IPC client V1)

Example Ejemplo: suscribirse a los mensajes

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {
```

```
    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                    new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                    SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                            streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to
topic: " + topic);
                } else {
                    throw e;
                }
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (InterruptedException e) {
```

```
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
                Optional.of(streamResponseHandler));
    }

    public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

        @Override
        public void onStreamEvent(IoTCoreMessage ioTCoreMessage) {
            try {
                String topic = ioTCoreMessage.getMessage().getTopicName();
                String message = new
String(ioTCoreMessage.getMessage().getPayload(),
                        StandardCharsets.UTF_8);
                System.out.printf("Received new message on topic %s: %s%n", topic,
message);
            } catch (Exception e) {
                System.err.println("Exception occurred while processing subscription
response " +
                        "message.");
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
```

```
            return false;
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to IoT Core stream closed.");
        }
    }
}
```

Python (IPC client V1)

Example Ejemplo: suscribirse a los mensajes

> **Note**
>
> En este ejemplo se supone que está utilizando la versión 1.5.4 o posterior de SDK para dispositivos con AWS IoT para Python v2.

```
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: IoTCoreMessage) -> None:
        try:
            message = str(event.message.payload, "utf-8")
            topic_name = event.message.topic_name
```

```
            # Handle message.
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        # Handle error.
        return True  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        # Handle close.
        pass


topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)

# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()
```

C++ (IPC client V1)

Example Ejemplo: suscribirse a los mensajes

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;
```

```
class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:
        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string topicName =
 message.value().GetTopicName().value().c_str();
                // Handle message.
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
```

```
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
```

```
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Ejemplo: suscribirse a los mensajes

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/
dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            const streamingOperation = this.ipcClient.subscribeToIoTCore(request);

            streamingOperation.on('message', (message: IoTCoreMessage) => {
```

```
                // parse the message depending on your use cases, e.g.
                if (message.message && message.message.payload) {
                    const receivedMessage = message.message.payload.toString();
                }
            });

            streamingOperation.on('streamError', (error : RpcError) => {
                // define your own error handling logic
            });

            streamingOperation.on('ended', () => {
                // define your own logic
            });

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToIoTCore = new SubscribeToIoTCore();
```

Rust

Example Ejemplo: suscribirse a los mensajes

```
use gg_sdk::{Qos, Sdk};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;

    let callback = |topic: &str, payload: &[u8]| {
        let message = String::from_utf8_lossy(payload);
        println!("Received new message on topic {topic}: {message}");
    };

    let _sub = sdk
        .subscribe_to_iot_core(topic, qos, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

C

Example Ejemplo: suscribirse a los mensajes

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle
```

```
) {
    (void) ctx;
    (void) handle;

    printf(
        "Received new message on topic %.*s: %.*s\n",
        (int) topic.len,
        topic.data,
        (int) payload.len,
        payload.data
    );
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_iot_core(
        topic, qos, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
```

```
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the subscription handle.
    ggipc_close_subscription(handle);
}
```

C++ (Component SDK)

Example Ejemplo: suscribirse a los mensajes

```
#include <gg/ipc/client.hpp>
#include <unistd.h>
#include <iostream>

class ResponseHandler : public gg::ipc::IotTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Buffer payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        std::cout << "Received new message on topic " << topic << ": "
                  << payload << "\n";
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    static ResponseHandler handler;
    error = client.subscribe_to_iot_core(topic, qos, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
```

```
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}
```

Ejemplos

Utilice los siguientes ejemplos para aprender a utilizar el servicio IPC de AWS IoT Core MQTT en sus componentes.

Ejemplo de editor AWS IoT Core MQTT (C++, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
```

```
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_iotcore_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
```

```
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio IPC de AWS IoT Core MQTT para publicar mensajes en. AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
```

```
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);

        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }
```

```
        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

Ejemplo de suscriptor AWS IoT Core MQTT (C++, cliente IPC V1)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      Run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
          Execute: OWNER
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio IPC de AWS IoT Core MQTT para suscribirse a los mensajes de. AWS IoT Core

```
#include <iostream>
```

```
#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:

        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string messageTopic =
 message.value().GetTopicName().value().c_str();
                std::cout << "Received new message on topic: " << messageTopic <<
 std::endl;
                std::cout << "Message: " << messageString << std::endl;
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to IoT Core stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
```

```
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
```

```
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

Ejemplo de editor AWS IoT Core MQTT (Rust)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherRust",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherRust:mqttproxy:1": {
```

```
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/publish_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherRust/1.0.0/publish_to_iot_core",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación Rust demuestra cómo utilizar el servicio IPC de AWS IoT Core MQTT para publicar mensajes en. AWS IoT Core

```
use gg_sdk::{Qos, Sdk};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let message = b"Hello, World";
    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;
```

```
    sdk.publish_to_iot_core(topic, message, qos)
        .expect("Failed to publish to topic");

    println!("Successfully published to topic: {topic}");
}
```

Ejemplo de suscriptor de AWS IoT Core MQTT (Rust)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberRust",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberRust:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/subscribe_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberRust/1.0.0/subscribe_to_iot_core",
          "Permission": {
```

```
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación Rust demuestra cómo utilizar el servicio IPC de AWS IoT Core MQTT para suscribirse a los mensajes de. AWS IoT Core

```
use gg_sdk::{Qos, Sdk};
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let topic = "my/topic";
    let qos = Qos::AtLeastOnce;

    let callback = |topic: &str, payload: &[u8]| {
        let message = String::from_utf8_lossy(payload);
        println!("Received new message on topic {topic}: {message}");
    };

    let _sub = sdk
        .subscribe_to_iot_core(topic, qos, &callback)
        .expect("Failed to subscribe to topic");

    println!("Successfully subscribed to topic: {topic}");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

Ejemplo de editor AWS IoT Core MQTT (C)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

```
{
```

```
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherC",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherC:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_publish_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherC/1.0.0/sample_publish_to_iot_core",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación en C muestra cómo utilizar el servicio IPC de AWS IoT Core MQTT para publicar mensajes en. AWS IoT Core

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer message = GG_STR("Hello, World");
    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    err = ggipc_publish_to_iot_core(topic, message, qos);
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to publish to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
    }

    printf(
        "Successfully published to topic: %.*s\n", (int) topic.len, topic.data
    );
}
```

Ejemplo de suscriptor AWS IoT Core MQTT (C)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberC",
  "ComponentVersion": "1.0.0",
```

```
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberC:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_subscribe_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberC/1.0.0/sample_subscribe_to_iot_core",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación en C muestra cómo utilizar el servicio IPC de AWS IoT Core MQTT para suscribirse a los mensajes de. AWS IoT Core

```
#include <gg/error.h>
#include <gg/ipc/client.h>
```

```
#include <gg/sdk.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
    void *ctx, GgBuffer topic, GgBuffer payload, GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    printf(
        "Received new message on topic %.*s: %.*s\n",
        (int) topic.len,
        topic.data,
        (int) payload.len,
        payload.data
    );
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer topic = GG_STR("my/topic");
    uint8_t qos = 1;

    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_iot_core(
        topic, qos, on_subscription_response, NULL, &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(
            stderr,
            "Failed to subscribe to topic: %.*s\n",
            (int) topic.len,
            topic.data
        );
        exit(-1);
```

```
    }

    printf(
        "Successfully subscribed to topic: %.*s\n", (int) topic.len, topic.data
    );

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the subscription handle.
    ggipc_close_subscription(handle);
}
```

Ejemplo de editor AWS IoT Core MQTT (C++, SDK de componentes)

La siguiente receta de ejemplo permite que el componente publique en todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": ["aws.greengrass#PublishToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
```

```
      "Lifecycle": {
        "run": "{artifacts:path}/sample_cpp_publish_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/sample_cpp_publish_to_iot_core",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio AWS IoT Core MQTT IPC para publicar mensajes en. AWS IoT Core

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view message = "Hello, World";
    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    error = client.publish_to_iot_core(topic, message, qos);
    if (error) {
        std::cerr << "Failed to publish to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully published to topic: " << topic << "\n";
}
```

Ejemplo de suscriptor de AWS IoT Core MQTT (C++, SDK de componentes)

La siguiente receta de ejemplo permite que el componente se suscriba a todos los temas.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": ["aws.greengrass#SubscribeToIoTCore"],
            "resources": ["*"]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "runtime": "*"
      },
      "Lifecycle": {
        "run": "{artifacts:path}/sample_cpp_subscribe_to_iot_core"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/sample_cpp_subscribe_to_iot_core",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
```

```
}
```

El siguiente ejemplo de aplicación de C++ demuestra cómo utilizar el servicio AWS IoT Core MQTT IPC para suscribirse a los mensajes de. AWS IoT Core

```
#include <gg/ipc/client.hpp>
#include <unistd.h>
#include <iostream>

class ResponseHandler : public gg::ipc::IotTopicCallback {
    void operator()(
        std::string_view topic,
        gg::Buffer payload,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        std::cout << "Received new message on topic " << topic << ": "
                  << payload << "\n";
    }
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view topic = "my/topic";
    uint8_t qos = 1;

    static ResponseHandler handler;
    error = client.subscribe_to_iot_core(topic, qos, handler);
    if (error) {
        std::cerr << "Failed to subscribe to topic: " << topic << "\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to topic: " << topic << "\n";

    // Keep the main thread alive, or the process will exit.
```

```
    while (1) {
        sleep(10);
    }
}
```

Interacción con el ciclo de vida del componente

Utilice el servicio IPC del ciclo de vida de los componentes para:

- Actualizar el estado del componente en el dispositivo principal.
- Suscribirse a las actualizaciones del estado de los componentes.
- Evitar que el núcleo detenga el componente para aplicar una actualización durante una implementación.
- Pausar y reanudar los procesos de los componentes.

Temas

- Versiones mínimas de SDK
- Autorización
- UpdateState
- SubscribeToComponentUpdates
- DeferComponentUpdate
- PauseComponent
- ResumeComponent

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para interactuar con el ciclo de vida de los componentes.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.2.10	

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.5.3	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0	
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	

Autorización

Para pausar o reanudar otros componentes de un componente personalizado, debe definir políticas de autorización que permitan a su componente administrar otros componentes. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización para la administración del ciclo de vida de los componentes tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.ipc.lifecycle`

Operación	Description (Descripción)	Recursos
`aws.greengrass#PauseComponent`	Permite que un componente detenga los componentes que especifique.	Un nombre de componente o * para permitir el acceso a todos los componentes.
`aws.greengrass#ResumeComponent`	Permite que un componente reanude los componentes que especifique.	Un nombre de componente o * para permitir el acceso a todos los componentes.
*	Permite que un component e pause y reanude los componentes que especifique.	Un nombre de componente o * para permitir el acceso a todos los componentes.

Ejemplos de políticas de autorización

Puede consultar el siguiente ejemplo de política de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo de política de autorización

El siguiente ejemplo de política de autorización permite a un componente pausar y reanudar todos los componentes.

```
{
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

UpdateState

Actualice el estado del componente en el dispositivo principal.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`state`

El estado que se va a establecer. Esta enumeración, `LifecycleState`, tiene los siguientes valores:

- `RUNNING`

- ERRORED

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Rust

Example Ejemplo: estado de actualización

```
use gg_sdk::{ComponentState, Sdk};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Update component state to RUNNING
    sdk.update_state(ComponentState::Running)
        .expect("Failed to update component state");

    println!("Successfully updated component state to RUNNING.");
}
```

C

Example Ejemplo: estado de actualización

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();
```

```
    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Update component state to RUNNING
    err = ggipc_update_state(GG_COMPONENT_STATE_RUNNING);
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to update component state.\n");
        exit(-1);
    }

    printf("Successfully updated component state to RUNNING.\n");
}
```

C++ (Component SDK)

Example Ejemplo: estado de actualización

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Update component state to RUNNING
    error = client.update_component_state(GG_COMPONENT_STATE_RUNNING);
    if (error) {
        std::cerr << "Failed to update component state.\n";
        exit(-1);
    }

    std::cout << "Successfully updated component state to RUNNING.\n";
}
```

SubscribeToComponentUpdates

Suscríbase para recibir notificaciones antes de que el software AWS IoT Greengrass principal actualice un componente. La notificación especifica si el núcleo se reiniciará o no como parte de la actualización.

El núcleo envía notificaciones de actualización solo si la política de actualización de componentes de la implementación especifica que se notifique a los componentes. El comportamiento predeterminado es notificar a los componentes. Para obtener más información, consulte Crear implementaciones el DeploymentComponentUpdatePolicyobjeto que puede proporcionar al llamar a la CreateDeploymentoperación.

> **Important**
>
> Las implementaciones locales no notifican a los componentes antes de las actualizaciones.

Esta es una operación de suscripción en la que se suscribe a un flujo de mensajes de eventos. Para usar esta operación, defina un identificador de respuesta de flujo con funciones que gestionen los mensajes de eventos, los errores y el cierre del flujo. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Tipo de mensaje del evento: `ComponentUpdatePolicyEvents`

> **Tip**
>
> Puede seguir un tutorial para aprender a desarrollar un componente que aplace condicionalmente las actualizaciones de los componentes. Para obtener más información, consulte Tutorial: Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes.

Solicitud

Esta solicitud de la operación no tiene parámetros.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`messages`

El flujo de mensajes de notificación. Este objeto, `ComponentUpdatePolicyEvents`, contiene la siguiente información:

`preUpdateEvent` (Python: `pre_update_event`)

(Opcional) Un evento que indica que el núcleo quiere actualizar un componente. Puede responder con la operación DeferComponentUpdate para confirmar o aplazar la actualización hasta que el componente esté listo para reiniciarse. Este objeto, `PreComponentUpdateEvent`, contiene la siguiente información:

`deploymentId` (Python: `deployment_id`)

El ID de la AWS IoT Greengrass implementación que actualiza el componente.

`isGgcRestarting` (Python: `is_ggc_restarting`)

Si el núcleo necesita reiniciarse o no después de aplicar la actualización.

`postUpdateEvent` (Python: `post_update_event`)

(Opcional) Un evento que indica que el núcleo actualizó un componente. Este objeto, `PostComponentUpdateEvent`, contiene la siguiente información:

`deploymentId` (Python: `deployment_id`)

El ID de la AWS IoT Greengrass implementación que actualizó el componente.

> **Note**
>
> Esta característica requiere la versión 2.7.0 o posterior del componente núcleo de Greengrass.

DeferComponentUpdate

Confirme o aplace la actualización de un componente que detecte con SubscribeToComponentUpdates. Debe especificar el tiempo que debe transcurrir antes de que el núcleo vuelva a comprobar si el componente está preparado para continuar con la actualización del componente. También puede utilizar esta operación para indicar al núcleo que su componente está listo para la actualización.

Si un componente no responde a la notificación de actualización del componente, el núcleo espera el tiempo que especifique en la política de actualización de componentes de la implementación. Transcurrido ese tiempo de espera, el núcleo continúa con la implementación. El tiempo de espera predeterminado de la actualización del componente es de 60 segundos. Para obtener más información, consulte Crear implementaciones y el DeploymentComponentUpdatePolicyobjeto que puede proporcionar al llamar a la CreateDeploymentoperación.

> **Tip**
>
> Puede seguir un tutorial para aprender a desarrollar un componente que aplace condicionalmente las actualizaciones de los componentes. Para obtener más información, consulte Tutorial: Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`deploymentId` (Python: `deployment_id`)

El identificador del AWS IoT Greengrass despliegue que se va a aplazar.

`message`

(Opcional) El nombre del componente cuyas actualizaciones se van a aplazar.

Toma el valor predeterminado del componente que realiza la solicitud.

`recheckAfterMs` (Python: `recheck_after_ms`)

El tiempo en milisegundos durante el que se debe aplazar la actualización. El núcleo espera esa cantidad de tiempo y luego envía otro `PreComponentUpdateEvent` que puede detectar con SubscribeToComponentUpdates.

Especifique `0` si desea confirmar la actualización. Esto indica al núcleo que el componente está listo para la actualización.

El valor predeterminado es cero milisegundos, lo que significa confirmar la actualización.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

PauseComponent

Esta función está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Pausa los procesos de un componente en el dispositivo principal. Para reanudar un componente, utilice la ResumeComponentoperación.

Solo puede pausar los componentes genéricos. Si intenta pausar cualquier otro tipo de componente, esta operación arroja un `InvalidRequestError`.

> **Note**
>
> Esta operación no puede pausar los procesos contenerizados, como los contenedores de Docker. Para pausar y reanudar un contenedor de Docker, puede usar los comandos docker pause y dockerunpause.

Esta operación no detiene las dependencias de los componentes ni los componentes que dependen del componente que se pausa. Tenga en cuenta este comportamiento al pausar un componente que es una dependencia de otro componente, ya que el componente dependiente puede tener problemas cuando su dependencia está en pausa.

Al reiniciar o apagar un componente en pausa, por ejemplo, durante una implementación, el núcleo de Greengrass reanuda el componente y ejecuta su ciclo de vida de apagado. Para obtener más información sobre reiniciar un componente, consulte RestartComponent.

> **Important**
>
> Para usar esta operación, debe definir una política de autorización que otorgue permiso para usar esta operación. Para obtener más información, consulte Autorización.

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para pausar y reanudar los componentes.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.4.3	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.6.2	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.13.1	
SDK para dispositivos con AWS IoT para JavaScript v2	versión 1.12.0	

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

El nombre del componente que se va a pausar, que debe ser un componente genérico. Para obtener más información, consulte Tipos de componentes.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

ResumeComponent

Esta función está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Reanuda los procesos de un componente en el dispositivo principal. Para pausar un componente, utilice la PauseComponentoperación.

Solo puede reanudar los componentes pausados. Si intenta reanudar un componente que no está en pausa, esta operación arroja un `InvalidRequestError`.

> **Important**
>
> Para utilizar esta operación, debe definir una política de autorización que conceda permiso para hacerlo. Para obtener más información, consulte Autorización.

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para pausar y reanudar los componentes.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.4.3	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.6.2	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.13.1	
SDK para dispositivos con AWS IoT para JavaScript v2	versión 1.12.0	

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

El nombre del componente a reanudar.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

Interacción con la configuración de componentes

El servicio IPC de configuración de componentes le permite hacer lo siguiente:

- Obtener y establecer los parámetros de configuración de los componentes.
- Suscribirse a las actualizaciones de configuración de los componentes.
- Validar las actualizaciones de configuración de los componentes antes de que el núcleo las aplique.

Temas

- Versiones mínimas de SDK
- GetConfiguration
- UpdateConfiguration
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates
- SendConfigurationValidityReport

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para interactuar con la configuración de los componentes.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.2.10	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.5.3	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0	

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	

GetConfiguration

Obtiene un valor de configuración para un componente del dispositivo principal. Usted especifica la ruta clave para la que se va a obtener un valor de configuración.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

(Opcional) El nombre del componente.

Toma el valor predeterminado del componente que realiza la solicitud.

`keyPath` (Python: `key_path`)

La ruta clave al valor de configuración. Especifique una lista en la que cada entrada sea la clave de un único nivel del objeto de configuración. Por ejemplo, especifique `["mqtt", "port"]` si desea obtener el valor de `port` en la siguiente configuración.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Para obtener la configuración completa del componente, especifique una lista vacía.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`componentName` (Python: `component_name`)

El nombre del componente.

`value`

La configuración solicitada como objeto.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Rust

Example Ejemplo: Obtener configuración

```
use core::mem::MaybeUninit;
use gg_sdk::{Sdk, UnpackedObject};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Get a configuration value at key path ["mqtt", "port"]
    let mut buf = [MaybeUninit::uninit(); 1024];

    let value = sdk
        .get_config(&["mqtt", "port"], None, &mut buf)
        .expect("Failed to get configuration");

    if let UnpackedObject::I64(port) = value.unpack() {
        println!("Configuration value: {port}");
    }
}
```

C

Example Ejemplo: Obtener la configuración

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <inttypes.h>
#include <stdio.h>
```

```
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Get a configuration value at key path ["mqtt", "port"]
    uint8_t response_mem[1024];
    GgObject value;

    err = ggipc_get_config(
        GG_BUF_LIST(GG_STR("mqtt"), GG_STR("port")),
        NULL, // component_name (NULL = current component)
        GG_BUF(response_mem),
        &value
    );
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to get configuration.\n");
        exit(-1);
    }

    if (gg_obj_type(value) == GG_TYPE_I64) {
        printf("Configuration value: %" PRId64 "\n", gg_obj_into_i64(value));
    } else if (gg_obj_type(value) == GG_TYPE_BUF) {
        GgBuffer buf = gg_obj_into_buf(value);
        printf("Configuration value: %.*s\n", (int) buf.len, buf.data);
    } else {
        printf("Configuration value is of unexpected type.\n");
    }
}
```

C++ (Component SDK)

Example Ejemplo: Obtener la configuración

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
```

```
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Get a configuration value at key path ["mqtt", "port"]
    std::array key_path = { gg::Buffer { "mqtt" }, gg::Buffer { "port" } };
    int64_t value = 0;

    error = client.get_config(key_path, std::nullopt, value);
    if (error) {
        std::cerr << "Failed to get configuration.\n";
        exit(-1);
    }

    std::cout << "Configuration value: " << value << "\n";
}
```

UpdateConfiguration

Actualiza un valor de configuración para este componente en el dispositivo principal.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`keyPath` (Python: `key_path`)

(Opcional) La ruta clave al nodo del contenedor (el objeto) que se va a actualizar. Especifique una lista en la que cada entrada sea la clave de un único nivel del objeto de configuración. Por ejemplo, especifique la ruta clave `["mqtt"]` y el valor de combinación `{ "port": 443 }` para establecer el valor `port` en la siguiente configuración.

```
{
  "mqtt": {
    "port": 443
  }
}
```

La ruta clave debe especificar un nodo del contenedor (un objeto) en la configuración. Si el nodo no existe en la configuración del componente, esta operación lo crea y establece su valor en el objeto de `valueToMerge`.

Toma el valor predeterminado raíz del objeto de configuración.

`timestamp`

Tiempo actual de Unix en milisegundos. Esta operación utiliza esta marca temporal para resolver las actualizaciones simultáneas de la clave. Si la clave de la configuración del componente tiene una marca temporal mayor que la marca temporal de la solicitud, la solicitud falla.

`valueToMerge` (Python: `value_to_merge`)

El objeto de configuración que se va a combinar en la ubicación que especifique en `keyPath`. Para obtener más información, consulte Actualización de las configuraciones de los componentes.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Rust

Example Ejemplo: actualizar la configuración

```
use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Update configuration value at key path ["mqtt", "port"] to 443
    sdk.update_config(&["mqtt", "port"], None, 443)
        .expect("Failed to update configuration");

    println!("Successfully updated configuration.");
```

```
}
```

C

Example Ejemplo: actualizar la configuración

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Update configuration value at key path ["mqtt", "port"] to 443
    err = ggipc_update_config(
        GG_BUF_LIST(GG_STR("mqtt"), GG_STR("port")),
        NULL, // timestamp (NULL = current time)
        gg_obj_i64(443)
    );
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to update configuration.\n");
        exit(-1);
    }

    printf("Successfully updated configuration.\n");
}
```

C++ (Component SDK)

Example Ejemplo: actualizar la configuración

```
#include <gg/ipc/client.hpp>
#include <iostream>
```

```
int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Update configuration value at key path ["mqtt", "port"] to 443
    std::array key_path = { gg::Buffer { "mqtt" }, gg::Buffer { "port" } };

    error = client.update_config(key_path, 443);
    if (error) {
        std::cerr << "Failed to update configuration.\n";
        exit(-1);
    }

    std::cout << "Successfully updated configuration.\n";
}
```

SubscribeToConfigurationUpdate

Suscríbase para recibir notificaciones cuando se actualice la configuración de un componente. Al suscribirse a una clave, recibirá una notificación cada vez que alguna de las claves secundarias se actualice.

Esta es una operación de suscripción en la que se suscribe a un flujo de mensajes de eventos. Para usar esta operación, defina un identificador de respuesta de flujo con funciones que gestionen los mensajes de eventos, los errores y el cierre del flujo. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Tipo de mensaje del evento: `ConfigurationUpdateEvents`

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

(Opcional) El nombre del componente.

Toma el valor predeterminado del componente que realiza la solicitud.

`keyPath` (Python: `key_path`)

La ruta clave al valor de configuración al que se va a suscribir. Especifique una lista en la que cada entrada sea la clave de un único nivel del objeto de configuración. Por ejemplo, especifique `["mqtt", "port"]` si desea obtener el valor de `port` en la siguiente configuración.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Para suscribirse a las actualizaciones de todos los valores de la configuración del componente, especifique una lista vacía.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`messages`

El flujo de mensajes de notificación. Este objeto, `ConfigurationUpdateEvents`, contiene la siguiente información:

`configurationUpdateEvent` (Python: `configuration_update_event`)

El evento de actualización de la configuración. Este objeto, `ConfigurationUpdateEvent`, contiene la siguiente información:

`componentName` (Python: `component_name`)

El nombre del componente.

`keyPath` (Python: `key_path`)

La ruta clave al valor de configuración que actualizó.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Rust

Example Ejemplo: suscríbase a las actualizaciones de configuración

```
use gg_sdk::Sdk;
use std::{thread, time::Duration};

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    // Subscribe to configuration updates for key path ["mqtt"]
    let callback = |component_name: &str, key_path: &[&str]| {
        println!(
            "Received configuration update for component: {component_name}"
        );
        println!("Key path: {key_path:?}");
    };

    let _sub = sdk
        .subscribe_to_configuration_update(None, &["mqtt"], &callback)
        .expect("Failed to subscribe to configuration updates");

    println!("Successfully subscribed to configuration updates.");

    // Keep the main thread alive, or the process will exit.
    loop {
        thread::sleep(Duration::from_secs(10));
    }
}
```

C

Example Ejemplo: suscríbase a las actualizaciones de configuración

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/object.h>
#include <gg/sdk.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void on_subscription_response(
```

```
    void *ctx,
    GgBuffer component_name,
    GgList key_path,
    GgIpcSubscriptionHandle handle
) {
    (void) ctx;
    (void) handle;

    printf(
        "Received configuration update for component: %.*s\n",
        (int) component_name.len,
        component_name.data
    );

    printf("Key path: [");
    for (size_t i = 0; i < key_path.len; i++) {
        if (i > 0) {
            printf(", ");
        }
        GgObject *obj = &key_path.items[i];
        if (gg_obj_type(*obj) == GG_TYPE_BUF) {
            GgBuffer key = gg_obj_into_buf(*obj);
            printf("\"%.*s\"", (int) key.len, key.data);
        }
    }
    printf("]\n");
}

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    // Subscribe to configuration updates for key path ["mqtt"]
    GgIpcSubscriptionHandle handle;
    err = ggipc_subscribe_to_configuration_update(
        NULL, // component_name (NULL = current component)
        GG_BUF_LIST(GG_STR("mqtt")),
        on_subscription_response,
        NULL,
```

```
        &handle
    );
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to subscribe to configuration updates.\n");
        exit(-1);
    }

    printf("Successfully subscribed to configuration updates.\n");

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }

    // To stop subscribing, close the stream.
    ggipc_close_subscription(handle);
}
```

C++ (Component SDK)

Example Ejemplo: suscríbase a las actualizaciones de configuración

```
#include <gg/ipc/client.hpp>
#include <unistd.h>
#include <iostream>

class ResponseHandler : public gg::ipc::ConfigurationUpdateCallback {
    void operator()(
        std::string_view component_name,
        gg::List key_path,
        gg::ipc::Subscription &handle
    ) override {
        (void) handle;
        std::cout << "Received configuration update for component: "
                  << component_name << "\n";
        std::cout << "Key path: [";
        for (size_t i = 0; i < key_path.size(); i++) {
            if (i > 0) {
                std::cout << ", ";
            }
            std::cout << "\"" << get<gg::Buffer>(key_path[i]) << "\"";
        }
        std::cout << "]\n";
    }
```

```
};

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    // Subscribe to configuration updates for key path ["mqtt"]
    std::array key_path = { gg::Buffer { "mqtt" } };

    static ResponseHandler handler;
    error = client.subscribe_to_configuration_update(
        key_path, std::nullopt, handler
    );
    if (error) {
        std::cerr << "Failed to subscribe to configuration updates.\n";
        exit(-1);
    }

    std::cout << "Successfully subscribed to configuration updates.\n";

    // Keep the main thread alive, or the process will exit.
    while (1) {
        sleep(10);
    }
}
```

SubscribeToValidateConfigurationUpdates

Suscríbase para recibir notificaciones antes de que se actualice la configuración de este componente. Esto permite a los componentes validar las actualizaciones de su propia configuración. Utilice la operación SendConfigurationValidityReport para indicar al núcleo si la configuración es válida o no.

> **Important**
>
> Las implementaciones locales no notifican a los componentes de las actualizaciones.

Esta es una operación de suscripción en la que se suscribe a un flujo de mensajes de eventos. Para usar esta operación, defina un identificador de respuesta de flujo con funciones que gestionen los mensajes de eventos, los errores y el cierre del flujo. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Tipo de mensaje del evento: `ValidateConfigurationUpdateEvents`

Solicitud

Esta solicitud de la operación no tiene parámetros.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`messages`

El flujo de mensajes de notificación. Este objeto, `ValidateConfigurationUpdateEvents`, contiene la siguiente información:

`validateConfigurationUpdateEvent` (Python: `validate_configuration_update_event`)

El evento de actualización de la configuración. Este objeto, `ValidateConfigurationUpdateEvent`, contiene la siguiente información:

`deploymentId` (Python: `deployment_id`)

El ID de la AWS IoT Greengrass implementación que actualiza el componente.

`configuration`

El objeto que contiene la nueva configuración.

SendConfigurationValidityReport

Indique al núcleo si una actualización de configuración de este componente es válida o no. La implementación falla si le dice al núcleo que la nueva configuración no es válida. Utilice la operación SubscribeToValidateConfigurationUpdates de suscripción para validar las actualizaciones de configuración.

Si un componente no responde a una notificación de esta validación, el núcleo espera el tiempo que especifique en la política de validación de la configuración de la implementación. Transcurrido ese

tiempo de espera, el núcleo continúa con la implementación. El tiempo de espera predeterminado de la validación del componente es de 20 segundos. Para obtener más información, consulte Crear implementaciones y el DeploymentConfigurationValidationPolicyobjeto que puede proporcionar al llamar a la CreateDeploymentoperación.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`configurationValidityReport` (Python: `configuration_validity_report`)

El informe que indica al núcleo si la actualización de configuración es válida o no. Este objeto, `ConfigurationValidityReport`, contiene la siguiente información:

`status`

El estado de validez. Esta enumeración, `ConfigurationValidityStatus`, tiene los siguientes valores:

- `ACCEPTED`: la configuración es válida y el núcleo puede aplicarla a este componente.
- `REJECTED`: la configuración no es válida y la implementación falla.

`deploymentId` (Python: `deployment_id`)

El ID de la AWS IoT Greengrass implementación que solicitó la actualización de la configuración.

`message`

(Opcional) Un mensaje que informa de los motivos por los que la configuración no es válida.

Respuesta

Esta operación no proporciona ninguna información en su respuesta.

Recupere valores secretos

Utilice el servicio IPC del administrador secreto para recuperar los valores secretos de los secretos del dispositivo principal. El componente administrador de secretos se utiliza para implementar secretos cifrados en los dispositivos principales. A continuación, puede utilizar una operación de IPC para descifrar el secreto y utilizar su valor en los componentes personalizados.

Temas

- Versiones mínimas de SDK
- Autorización
- GetSecretValue
- Ejemplos

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para recuperar los valores secretos de los secretos del dispositivo principal.

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Java v2	Versión 1.2.10
SDK para dispositivos con AWS IoT para Python v2	Versión 1.5.3
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0

Autorización

Para usar el administrador de secretos en un componente personalizado, debe definir políticas de autorización que permitan a su componente obtener el valor de los secretos que almacena en el dispositivo principal. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización del administrador de secretos tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.SecretManager`

Operación	Description (Descripción)	Recursos
`aws.greengrass#GetSecretValue`	Permite que un componente obtenga el valor de los secretos que están cifrados en el dispositivo principal.	Un ARN secreto de Secrets Manager, o * para permitir el acceso a todos los secretos.

Ejemplos de políticas de autorización

Puede consultar el siguiente ejemplo de política de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo de política de autorización

El siguiente ejemplo de política de autorización permite a un componente obtener el valor de cualquier secreto del dispositivo principal.

> **Note**
>
> En un entorno de producción, se recomienda reducir el alcance de la política de autorización para que el componente recupere solo los secretos que utiliza. Puede cambiar el * comodín por una lista de secretos ARNs al implementar el componente.

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

Obtiene el valor de un secreto que se almacena en el dispositivo principal.

Esta operación es similar a la operación de Secrets Manager, que puede utilizar para obtener el valor de un secreto en Nube de AWS. Para obtener más información, consulta GetSecretValue en la AWS Secrets Manager Referencia de la API de .

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`refresh` (Python: `refresh`)

(opcional): si se debe sincronizar el secreto solicitado con su último valor del AWS Secrets Manager servicio.

Si se establece en true, el administrador de secretos solicitará al AWS Secrets Manager servicio el valor más reciente de la etiqueta secreta especificada y devolverá ese valor como respuesta. De lo contrario, se devolverá el valor secreto que estaba almacenado localmente.

Este parámetro no funcionará junto con el parámetro `versionId` en la solicitud. Este parámetro funciona cuando se utiliza junto con la versión 2.13.0 y posteriores del núcleo.

`secretId` (Python: `secret_id`)

El nombre del secreto que se obtendrá. Puede especificar el Nombre de recurso de Amazon (ARN) o el nombre fácil de recordar del secreto.

`versionId` (Python: `version_id`)

(Opcional) El ID de versión que se obtendrá.

Puede especificar `versionId` o `versionStage`.

Si no especifica `versionId` o `versionStage`, esta operación se establece de forma predeterminada en la versión con la etiqueta `AWSCURRENT`.

`versionStage` (Python: `version_stage`)

(Opcional) La etiqueta de fase de la versión que se obtendrá.

Puede especificar `versionId` o `versionStage`.

Si no especifica `versionId` o `versionStage`, esta operación se establece de forma predeterminada en la versión con la etiqueta `AWSCURRENT`.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`secretId` (Python: `secret_id`)

El ID del secreto.

`versionId` (Python: `version_id`)

El ID de esta versión del secreto.

`versionStage` (Python: `version_stage`)

La lista de etiquetas de fase adjunta a la versión del secreto.

`secretValue` (Python: `secret_value`)

El valor de esta versión del secreto. Este objeto, `SecretValue`, contiene la siguiente información.

`secretString` (Python: `secret_string`)

La parte descifrada de la información secreta protegida que proporcionó a Secrets Manager en forma de cadena.

`secretBinary` (Python: `secret_binary`)

(Opcional) La parte descifrada de la información secreta protegida que proporcionó a Secrets Manager como datos binarios en forma de matriz de bytes. Esta propiedad contiene los datos binarios como una cadena codificada en base64.

Esta propiedad no se utiliza si creó el secreto en la consola de Secrets Manager.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V1)

Example Ejemplo: obtener un valor secreto

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String secretArn = args[0];
        String versionStage = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetSecretValueResponseHandler responseHandler =
                    GetSecretValue.getSecretValue(ipcClient, secretArn,
 versionStage);
            CompletableFuture<GetSecretValueResponse> futureResponse =
```

```
                    responseHandler.getResponse();
            try {
                GetSecretValueResponse response =
 futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                response.getSecretValue().postFromJson();
                String secretString = response.getSecretValue().getSecretString();
                System.out.println("Successfully retrieved secret value: " +
 secretString);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while retrieving secret: " +
 secretArn);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while retrieving secret:
 " + secretArn);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetSecretValueResponseHandler
 getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
 String versionStage) {
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Ejemplo: obtener un valor secreto

> **Note**
>
> En este ejemplo se supone que está utilizando la versión 1.5.4 o posterior de SDK para dispositivos con AWS IoT para Python v2.

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Example Ejemplo: obtener un valor secreto

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
```

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
    private readonly secretId : string;
    private readonly versionStage : string;
    private ipcClient : greengrasscoreipc.Client

    constructor() {
        this.secretId = "<define_your_own_secretId>"
        this.versionStage = "<define_your_own_versionStage>"

        this.getSecretValue().then(r => console.log("Started workflow"));
    }

    private async getSecretValue() {
        try {
            this.ipcClient = await getIpcClient();

            const getSecretValueRequest : GetSecretValueRequest = {
                secretId: this.secretId,
                versionStage: this.versionStage,
            };

            const result = await
 this.ipcClient.getSecretValue(getSecretValueRequest);
            const secretString = result.secretValue.secretString;
            console.log("Successfully retrieved secret value: " + secretString)
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
```

```
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

Ejemplos

Utilice los siguientes ejemplos para aprender a utilizar el servicio IPC del administrador de secretos en sus componentes.

Ejemplo: impresión del secreto (Python, cliente IPC V1)

Este componente de ejemplo imprime el valor de un secreto que se implementa en el dispositivo principal.

> **Important**
>
> Este componente de ejemplo imprime el valor de un secreto, así que utilícelo solo con los secretos que almacenan datos de prueba. No utilice este componente para imprimir el valor de un secreto que almacena información importante.

Temas

- Fórmula
- Artefactos
- De uso

Fórmula

La siguiente receta de ejemplo define un parámetro de configuración de ARN oculto y permite que el componente obtenga el valor de cualquier secreto del dispositivo principal.

> **Note**
>
> En un entorno de producción, se recomienda reducir el alcance de la política de autorización para que el componente recupere solo los secretos que utiliza. Puede cambiar el * comodín por una lista de secretos ARNs al implementar el componente.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "Run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "Run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
```

```
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      Run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      Run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
```

Artefactos

El siguiente ejemplo de aplicación de Python demuestra cómo utilizar el servicio IPC del administrador de secretos para obtener el valor de un secreto en el dispositivo principal.

```
import concurrent.futures
import json
import sys
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
```

```
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()

    try:
        response = future_response.result(TIMEOUT)
        secret_json = json.loads(response.secret_value.secret_string)
        print('Successfully got secret: ' + secret_id)
        print('Secret value: ' + str(secret_json))
    except concurrent.futures.TimeoutError:
        print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
    except UnauthorizedError as e:
        print('Unauthorized error while getting secret: ' + secret_id,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while getting secret: ' + secret_id, file=sys.stderr)
        raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

De uso

Puede usar este componente de ejemplo con el componente de administrador de secretos para implementar e imprimir el valor de un secreto en su dispositivo principal.

Cómo crear, implementar e imprimir un secreto de prueba

1. Crear un secreto en Secrets Manager con datos de prueba.

   Linux or Unix

   ```
   aws secretsmanager create-secret \
     --name MyTestGreengrassSecret \
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Windows Command Prompt (CMD)

   ```
   aws secretsmanager create-secret ^
     --name MyTestGreengrassSecret ^
   ```

```
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

PowerShell

```
aws secretsmanager create-secret `
  --name MyTestGreengrassSecret `
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

Guarde el ARN del secreto para utilizarlo en los pasos siguientes.

Para obtener más información, consulte Creación de un secreto en la Guía del usuario de AWS Secrets Manager .

2. Implemente el componente de administrador de secretos (`aws.greengrass.SecretManager`) con la siguiente actualización de combinación de configuraciones. Especifique el ARN del secreto que creó anteriormente.

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos o el comando de implementación de la CLI de Greengrass.

3. Cree e implemente el componente de ejemplo de esta sección con la siguiente actualización de combinación de configuraciones. Especifique el ARN del secreto que creó anteriormente.

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
```

```
        ],
        "resources": [
          "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
        ]
      }
    }
  }
}
```

Para obtener más información, consulte Creación de componentes de AWS IoT Greengrass

4. Consulte los registros del software AWS IoT Greengrass principal para comprobar que las implementaciones se han realizado correctamente y consulte el registro de `com.example.PrintSecret` componentes para ver impreso el valor secreto. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Interactúe con las sombras locales

Utilice el servicio IPC de sombra para interactuar con las sombras locales de un dispositivo. El dispositivo con el que elija interactuar puede ser su dispositivo principal o un dispositivo de cliente conectado.

Para utilizar estas operaciones de IPC, incluya el componente administrador de sombras como una dependencia en su componente personalizado. A continuación, puede utilizar las operaciones de IPC en sus componentes personalizados para interactuar con las sombras locales del dispositivo a través del administrador de sombras. Para permitir que los componentes personalizados reaccionen a los cambios en los estados paralelos locales, también puede utilizar el servicio publish/subscribe IPC para suscribirse a eventos paralelos. Para obtener más información sobre el uso del publish/subscribe servicio, consulte laPublicar/suscribir mensajes locales.

> **Note**
>
> Para permitir que un dispositivo principal interactúe con las sombras de dispositivos de cliente, también debe configurar e implementar el componente puente MQTT. Para obtener más información, consulte Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente.

Temas

- Versiones mínimas de SDK
- Autorización
- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas del SDK para dispositivos con AWS IoT que debe utilizar para interactuar con las sombras locales.

SDK	Versión mínima
SDK para dispositivos con AWS IoT para Java v2	v1.4.0
SDK para dispositivos con AWS IoT para Python v2	Versión 1.6.0
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0

Autorización

Para utilizar el servicio IPC de sombra en un componente personalizado, debe definir políticas de autorización que permitan a su componente interactuar con las sombras. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización para la interacción con las sombra tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.ShadowManager`

Operación	Description (Descripción)	Recursos
`aws.greengrass#GetThingShadow`	Permite que un componente recupere la sombra de un objeto.	Una de las siguientes cadenas: • `$aws/things/`*`thingName`*`/shadow/`, para permitir el acceso a la sombra de dispositivo clásico. • `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*, para permitir el acceso a una sombra con nombre. • `*`, para permitir el acceso a todas las sombras.
`aws.greengrass#UpdateThingShadow`	Permite que un componente actualice la sombra de un objeto.	Una de las siguientes cadenas: • `$aws/things/`*`thingName`*`/shadow/`, para permitir el acceso a la sombra de dispositivo clásico. • `$aws/things/`*`thingName`*`/shadow/name/`*`shadowName`*, para permitir el acceso a una sombra con nombre. • `*`, para permitir el acceso a todas las sombras.

Operación	Description (Descripción)	Recursos
`aws.greengrass#DeleteThingShadow`	Permite a un componente eliminar la sombra de un objeto.	Una de las siguientes cadenas: • `$aws/things/` *`thingName`* `/shadow/`, para permitir el acceso a la sombra de dispositivo clásico. • `$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`* , para permitir el acceso a una sombra con nombre. • `*`, para permitir el acceso a todas las sombras.
`aws.greengrass#ListNamedShadowsForThing`	Permite que un componente recupere la lista de sombras con nombre de un objeto.	Cadena con el nombre de un objeto que permite acceder al objeto para enumerar sus sombras. Se usa `*` para permitir el acceso a todos los objetos.

Identificador de servicio IPC: `aws.greengrass.ipc.pubsub`

Operación	Description (Descripción)	Recursos
`aws.greengrass#SubscribeToTopic`	Permite que un componente se suscriba a los mensajes de los temas que especifique.	Una de las siguientes cadenas de temas: • *`shadowTopicPrefix`* `/get/accepted`

Operación	Description (Descripción)	Recursos
		• *shadowTopicPrefix* / get/rejected • *shadowTopicPrefix* / delete/accepted • *shadowTopicPrefix* / delete/rejected • *shadowTopicPrefix* / update/accepted • *shadowTopicPrefix* / update/delta • *shadowTopicPrefix* / update/rejected El valor del prefijo del tema *shadowTopicPrefix* depende del tipo de sombra: • Sombra clásica: $aws/things/ *thingName* /shadow • Sombra con nombre: $aws/things/ *thingName* /shadow/name/ *shadowName* Se usa * para permitir el acceso a todos los temas. En el núcleo de Greengrass versión 2.6.0 y versiones posteriores, puede suscribirse a temas que contengan comodines de temas MQTT

Operación	Description (Descripción)	Recursos
		(# y +). Esta cadena de tema admite los comodines de los temas MQTT como caractere s literales. Por ejemplo, si la política de autorización de un componente permite el acceso a `test/topic/#`, el componente se puede suscribir a `test/topic/#` , pero no se puede suscribir a `test/topic/filter` .

Variables de receta en las políticas de autorización de sombras locales

Si usa la versión 2.6.0 o posterior del núcleo de Greengrass y establece la opción de interpolateComponentConfigurationconfiguración del núcleo de Greengrass en`true`, puede usar la variable de receta en las políticas de autorización. `{iot:thingName}` Esta característica le permite configurar una política de autorización única para un grupo de dispositivos principales, de forma que cada dispositivo principal solo pueda acceder a su propia sombra. Por ejemplo, puede permitir que un componente acceda al siguiente recurso para realizar operaciones de IPC de sombra.

```
$aws/things/{iot:thingName}/shadow/
```

Ejemplos de políticas de autorización

Puede consultar los siguientes ejemplos de políticas de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo: permitir que un grupo de dispositivos principales interactúe con las sombras locales

> **Important**
>
> En este ejemplo, se utiliza una característica que está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. El núcleo de Greengrass

versión 2.6.0 suma compatibilidad con la mayoría de las variables de receta, por ejemplo: `{iot:thingName}`, en las configuraciones de componentes. Para activar esta función, defina la opción de interpolateComponentConfigurationconfiguración del núcleo de Greengrass en. `true` Si desea ver un ejemplo que funcione en todas las versiones del núcleo de Greengrass, consulte el ejemplo de política de autorización para un dispositivo de un núcleo principal.

El siguiente ejemplo de política de autorización permite que el componente `com.example.MyShadowInteractionComponent` interactúe con la sombra de dispositivo clásico y con la sombra con nombre `myNamedShadow` del dispositivo principal que ejecuta el componente. Esta política también permite que este componente reciba mensajes sobre temas locales relacionados con estas sombras.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
```

```
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/get/accepted",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted
```

Example Ejemplo: permitir que un grupo de dispositivos principales interactúe con sombra de dispositivo del cliente

> **Important**
>
> Esta característica requiere la versión 2.6.0 o versiones posteriores del núcleo de Greengrass, la versión 2.2.0 y versiones posteriores del administrador de sombras y la versión 2.2.0 o versiones posteriores del puente de MQTT. Debe configurar el puente MQTT para permitir que el administrador de sombra se comunique con los dispositivos de cliente.

El siguiente ejemplo de política de autorización permite que el componente `com.example.MyShadowInteractionComponent` interactúe con todas las sombras de dispositivos de los dispositivos de cliente cuyos nombres comiencen con `MyClientDevice`.

> **Note**
>
> Para permitir que un dispositivo principal interactúe con las sombras de dispositivos de cliente, también debe configurar e implementar el componente puente MQTT. Para obtener más información, consulte Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
```

```
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyClientDevice*"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*
```

Example Ejemplo: permitir que un solo dispositivo principal interactúe con las sombras locales

El siguiente ejemplo de política de autorización permite que el componente `com.example.MyShadowInteractionComponent` interactúe con la sombra de dispositivo clásico y con la sombra con nombre `myNamedShadow` para el dispositivo `MyThingName`. Esta política también permite que este componente reciba mensajes sobre temas locales relacionados con estas sombras.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted
```

Example Ejemplo: permitir que un grupo de dispositivos principales reaccione a los cambios en el estado de la sombra local

> **Important**
>
> En este ejemplo, se utiliza una característica que está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass. El núcleo de Greengrass versión 2.6.0 suma compatibilidad con la mayoría de las variables de receta, por ejemplo: `{iot:thingName}`, en las configuraciones de componentes. Para activar esta función, defina la opción de interpolateComponentConfigurationconfiguración del núcleo de Greengrass en. `true` Si desea ver un ejemplo que funcione en todas las versiones del

núcleo de Greengrass, consulte el ejemplo de política de autorización para un dispositivo de un núcleo principal.

El siguiente ejemplo de política de control de acceso permite personalizar `com.example.MyShadowReactiveComponent` para recibir mensajes sobre el tema `/update/delta` de la sombra de dispositivo clásico y de la sombra con nombre `myNamedShadow` en cada dispositivo principal en el que se ejecute el componente.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Example Ejemplo: permitir que un solo dispositivo principal reaccione a los cambios en el estado de la sombra local

El siguiente ejemplo de política de control de acceso permite personalizar `com.example.MyShadowReactiveComponent` para recibir mensajes sobre el tema `/update/delta` de la sombra de dispositivo clásico y de la sombra con nombre `myNamedShadow` para el dispositivo `MyThingName`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/update/delta
        - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

Obtenga la sombra de objeto especificado.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`thingName` (Python: `thing_name`)

El nombre del objeto.

Tipo: `string`

`shadowName` (Python: `shadow_name`)

El nombre de la sombra. Para especificar la sombra clásica del objeto, defina este parámetro en una cadena vacía (`""`).

> **Warning**
>
> El AWS IoT Greengrass servicio usa el `AWSManagedGreengrassV2Deployment` nombre shadow para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

Tipo: `string`

Respuesta

Esta respuesta de operación contiene la siguiente información:

`payload`

El documento de estado de la respuesta en forma de blob.

Tipo: `object` que contiene la siguiente información:

`state`

La información del estado.

Este objeto contiene la siguiente información:

`desired`

Las propiedades y los valores de estado que se solicitan actualizar en el dispositivo.

Tipo: `map` de pares clave-valor

`reported`

Las propiedades y los valores de estado informados por el dispositivo.

Tipo: `map` de pares clave-valor

`delta`

La diferencia entre las propiedades y los valores de estado deseados e informados. Esta propiedad solo está presente si los estados `desired` y `reported` son diferentes.

Tipo: `map` de pares clave-valor

`metadata`

Las marcas temporales de cada atributo de las secciones `desired` y `reported` para que se pueda determinar cuándo se actualizó el estado.

Tipo: `string`

`timestamp`

La fecha y hora de inicio en que se generó la respuesta.

Tipo: `integer`

`clientToken` (Python: `clientToken`)

El token que se usa para hacer coincidir la solicitud y la respuesta correspondiente.

Tipo: `string`

`version`

La versión del documento de sombra local.

Tipo: `integer`

Errores

Esta operación puede devolver los siguientes errores.

`InvalidArgumentsError`

El servicio de sombra local no puede validar los parámetros de la solicitud. Esto puede ocurrir si la solicitud contiene un formato incorrecto de JSON o caracteres no admitidos.

`ResourceNotFoundError`

No se encuentra el documento de sombra local solicitado.

`ServiceError`

Se ha producido un error de servicio interno o la cantidad de solicitudes al servicio de IPC ha superado los límites especificados en los parámetros de configuración de `maxLocalRequestsPerSecondPerThing` y `maxTotalLocalRequestsRate` del componente del administrador de sombra.

`UnauthorizedError`

La política de autorización del componente no incluye los permisos necesarios para esta operación.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V1)

Example Ejemplo: obtener la sombra de un objeto

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;
```

```
import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                    GetThingShadow.getThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
 futureResponse.get(TIMEOUT_SECONDS,
                        TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
 StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
 shadowName,
                        shadowPayload);
            } catch (TimeoutException e) {
```

```
                System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to get: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetThingShadowResponseHandler
 getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
 String shadowName) {
        GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
        getThingShadowRequest.setThingName(thingName);
        getThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Ejemplo: obtener la sombra de un objeto

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest
```

```
TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName

        # retrieve the GetThingShadow response after sending the request to the IPC
 server
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Ejemplo: obtener la sombra de un objeto

```
import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
```

```
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleGetThingShadowOperation(this.thingName,
                this.shadowName);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleGetThingShadowOperation(
        thingName: string,
        shadowName: string
    ) {
        const request: GetThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        };
        const response = await this.ipcClient.getThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use caseså
        throw err
```

```
    }
}

const startScript = new GetThingShadow();
```

UpdateThingShadow

Actualice la sombra del objeto especificado. Si la sombra no existe, se crea una.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`thingName` (Python: `thing_name`)

El nombre del objeto.

Tipo: `string`

`shadowName` (Python: `shadow_name`)

El nombre de la sombra. Para especificar la sombra clásica del objeto, defina este parámetro en una cadena vacía ("").

> **Warning**
>
> El AWS IoT Greengrass servicio usa el `AWSManagedGreengrassV2Deployment` nombre shadow para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

Tipo: `string`

`payload`

El documento del estado de solicitud es un blob.

Tipo: `object` que contiene la siguiente información:

`state`

La información del estado que se va a actualizar. Esta operación de IPC afecta únicamente a los campos especificados.

Este objeto contiene la siguiente información: Normalmente, utilizará la propiedad `desired` o la propiedad `reported`, pero no ambas en la misma solicitud.

`desired`

Las propiedades y los valores de estado que se solicitan actualizar en el dispositivo.

Tipo: `map` de pares clave-valor

`reported`

Las propiedades y los valores de estado informados por el dispositivo.

Tipo: `map` de pares clave-valor

`clientToken` (Python: `client_token`)

(Opcional) El token que se usa para hacer coincidir la solicitud y la respuesta correspondiente con el token del cliente.

Tipo: `string`

`version`

(Opcional) La versión del documento de sombra local que se va a actualizar. El servicio de sombra procesa la actualización solo si la versión especificada coincide con la versión más reciente que tiene.

Tipo: `integer`

Respuesta

Esta respuesta de operación contiene la siguiente información:

`payload`

El documento de estado de la respuesta en forma de blob.

Tipo: `object` que contiene la siguiente información:

`state`

La información del estado.

Este objeto contiene la siguiente información:

`desired`

Las propiedades y los valores de estado que se solicitan actualizar en el dispositivo.

Tipo: `map` de pares clave-valor

`reported`

Las propiedades y los valores de estado informados por el dispositivo.

Tipo: `map` de pares clave-valor

`delta`

Las propiedades y los valores de estado informados por el dispositivo.

Tipo: `map` de pares clave-valor

`metadata`

Las marcas temporales de cada atributo de las secciones `desired` y `reported` para que se pueda determinar cuándo se actualizó el estado.

Tipo: `string`

`timestamp`

La fecha y hora de inicio en que se generó la respuesta.

Tipo: `integer`

`clientToken` (Python: `client_token`)

El token que se usa para hacer coincidir la solicitud y la respuesta correspondiente.

Tipo: `string`

`version`

La versión del documento de sombra local una vez finalizada la actualización.

Tipo: `integer`

Errores

Esta operación puede devolver los siguientes errores.

`ConflictError`

El servicio de sombra local detectó un conflicto de versiones durante la operación de actualización. Esto ocurre cuando la versión de la carga útil de la solicitud no coincide con la versión del último documento de sombra local disponible.

`InvalidArgumentsError`

El servicio de sombra local no puede validar los parámetros de la solicitud. Esto puede ocurrir si la solicitud contiene un formato incorrecto de JSON o caracteres no admitidos.

Un `payload` válido incluye las siguientes propiedades:

- El nodo `state` existe y es un objeto que contiene la información de estado `desired` o `reported`.
- Los nodos `desired` y `reported` son objetos o nulos. Al menos uno de estos objetos debe contener información de estado válida.
- La profundidad de los objetos `desired` y `reported` no puede superar los ocho nodos.
- La longitud del valor `clientToken` no puede superar los 64 caracteres.
- El valor `version` debe ser 1 o superior.

`ServiceError`

Se ha producido un error de servicio interno o la cantidad de solicitudes al servicio de IPC ha superado los límites especificados en los parámetros de configuración de `maxLocalRequestsPerSecondPerThing` y `maxTotalLocalRequestsRate` del componente del administrador de sombra.

`UnauthorizedError`

La política de autorización del componente no incluye los permisos necesarios para esta operación.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V1)

Example Ejemplo: actualizar una sombra de objeto

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
```

```
            UpdateThingShadowResponseHandler responseHandler =
                    UpdateThingShadow.updateThingShadow(ipcClient, thingName,
 shadowName,
                            shadowPayload);
            CompletableFuture<UpdateThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully updated shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while updating shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while updating shadow: %s/
%s%n",
                            thingName, shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
 updateThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
 UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCClient.updateThingShadow(updateThingShadowRequest,
                Optional.empty());
    }
```

```
}
```

Python (IPC client V1)

Example Ejemplo: actualizar una sombra de objeto

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Example Ejemplo: actualizar una sombra de objeto

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
```

```
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;
    private shadowDocumentStr: string;

    constructor() {
        // Define args parameters here

        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.shadowDocumentStr = "<define_your_own_payload>";

        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleUpdateThingShadowOperation(
                this.thingName,
                this.shadowName,
                this.shadowDocumentStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleUpdateThingShadowOperation(
        thingName: string,
        shadowName: string,
        payloadStr: string
    ) {
        const request: UpdateThingShadowRequest = {
            thingName: thingName,
```

```
            shadowName: shadowName,
            payload: payloadStr
        }
        // make the UpdateThingShadow request
        const response = await this.ipcClient.updateThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();
```

DeleteThingShadow

Elimina la sombra de objeto especificado.

A partir de la versión 2.0.4 de administrador de sombra, cuando se elimina una sombra se incrementa el número de versión. Por ejemplo, si elimina la sombra `MyThingShadow` en la versión 1, la versión de la sombra eliminada es 2. Si después vuelve a crear una sombra con el nombre `MyThingShadow`, la versión de esa sombra es 3.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`thingName` (Python: `thing_name`)

El nombre del objeto.

Tipo: `string`

`shadowName` (Python: `shadow_name`)

El nombre de la sombra. Para especificar la sombra clásica del objeto, defina este parámetro en una cadena vacía ("").

> **Warning**
>
> El AWS IoT Greengrass servicio usa el `AWSManagedGreengrassV2Deployment` nombre shadow para administrar las implementaciones que se dirigen a dispositivos principales individuales. Esta sombra denominada está reservada para que la utilice el AWS IoT Greengrass servicio. No actualice ni elimine esta sombra con nombre.

Tipo: `string`

Respuesta

Esta respuesta de operación contiene la siguiente información:

`payload`

Un documento de estado de respuesta vacío.

Errores

Esta operación puede devolver los siguientes errores.

`InvalidArgumentsError`

El servicio de sombra local no puede validar los parámetros de la solicitud. Esto puede ocurrir si la solicitud contiene un formato incorrecto de JSON o caracteres no admitidos.

`ResourceNotFoundError`

No se encuentra el documento de sombra local solicitado.

`ServiceError`

Se ha producido un error de servicio interno o la cantidad de solicitudes al servicio de IPC ha superado los límites especificados en los parámetros de configuración de

`maxLocalRequestsPerSecondPerThing` y `maxTotalLocalRequestsRate` del componente del administrador de sombra.

`UnauthorizedError`

La política de autorización del componente no incluye los permisos necesarios para esta operación.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V1)

Example Ejemplo: eliminar una sombra de objeto

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {
```

```
    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            DeleteThingShadowResponseHandler responseHandler =
                    DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<DeleteThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to delete: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
```

```
        }
    }

    public static DeleteThingShadowResponseHandler
 deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName) {
        DeleteThingShadowRequest deleteThingShadowRequest = new
 DeleteThingShadowRequest();
        deleteThingShadowRequest.setThingName(thingName);
        deleteThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example Ejemplo: eliminar una sombra de objeto

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName

        # retrieve the DeleteThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_delete_thing_shadow()
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload
```

```
    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Ejemplo: eliminar una sombra de objeto

```
import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleDeleteThingShadowOperation(this.thingName,
 this.shadowName)
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {
```

```
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

Enumera las sombras con nombre del objeto especificado.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`thingName` (Python: `thing_name`)

El nombre del objeto.

Tipo: `string`

`pageSize` (Python: `page_size`)

(Opcional) La cantidad de nombres de sombra que se devuelve en cada llamada.

Tipo: `integer`

Predeterminado: 25

Máximo: 100

`nextToken` (Python: `next_token`)

(Opcional) El token para recuperar el siguiente grupo de resultados. Este valor se devuelve en los resultados paginados y se utiliza en la llamada que devuelve la página siguiente.

Tipo: `string`

Respuesta

Esta respuesta de operación contiene la siguiente información:

`results`

La lista de sombras con nombre.

Tipo: `array`

`timestamp`

(Opcional) La fecha y hora en que se generó la respuesta.

Tipo: `integer`

`nextToken` (Python: `next_token`)

(Opcional) El valor del token que se utilizará en las solicitudes paginadas para recuperar la siguiente página de la secuencia. Este token no está presente cuando no hay más sombras con nombre que devolver.

Tipo: `string`

> **Note**
>
> Si el tamaño de página solicitado coincide exactamente con la cantidad de nombres de sombra de la respuesta, entonces este token está presente; sin embargo, cuando se usa, devuelve una lista vacía.

Errores

Esta operación puede devolver los siguientes errores.

`InvalidArgumentsError`

El servicio de sombra local no puede validar los parámetros de la solicitud. Esto puede ocurrir si la solicitud contiene un formato incorrecto de JSON o caracteres no admitidos.

`ResourceNotFoundError`

No se encuentra el documento de sombra local solicitado.

`ServiceError`

Se ha producido un error de servicio interno o la cantidad de solicitudes al servicio de IPC ha superado los límites especificados en los parámetros de configuración de `maxLocalRequestsPerSecondPerThing` y `maxTotalLocalRequestsRate` del componente del administrador de sombra.

`UnauthorizedError`

La política de autorización del componente no incluye los permisos necesarios para esta operación.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Java (IPC client V1)

Example Ejemplo: enumerar las sombras con nombre de un objeto

> **Note**
>
> En este ejemplo, se utiliza una `IPCUtils` clase para crear una conexión con el servicio AWS IoT Greengrass Core IPC. Para obtener más información, consulte Conéctese al AWS IoT Greengrass servicio Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;
```

```
import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
 software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
import
 software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
 response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

 ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                    nextToken, 25);
                    CompletableFuture<ListNamedShadowsForThingResponse>
 futureResponse =
                            responseHandler.getResponse();
```

```
                    ListNamedShadowsForThingResponse response =
                            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                    List<String> responseNamedShadows = response.getResults();
                    namedShadows.addAll(responseNamedShadows);
                    nextToken = response.getNextToken();
                } while (nextToken != null);
                System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
                        String.join(",", namedShadows));
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while listing named shadows for
 thing: " + thingName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while listing named
 shadows for " +
                            "thing: " + thingName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.println("Unable to find thing to list named shadows:
 " + thingName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static ListNamedShadowsForThingResponseHandler
 listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String nextToken, int pageSize) {
        ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
                new ListNamedShadowsForThingRequest();
        listNamedShadowsForThingRequest.setThingName(thingName);
        listNamedShadowsForThingRequest.setNextToken(nextToken);
        listNamedShadowsForThingRequest.setPageSize(pageSize);
        return
 greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
                Optional.empty());
```

```
    }
}
```

Python (IPC client V1)

Example Ejemplo: enumerar las sombras con nombre de un objeto

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10

def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize

        # retrieve the ListNamedShadowsForThingRequest response after sending the
 request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()

        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Ejemplo: enumerar las sombras con nombre de un objeto

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
```

```
    ) {
        let request: ListNamedShadowsForThingRequest = {
            thingName: thingName,
            nextToken: nextToken,
        };
        if (pageSizeStr) {
            request.pageSize = parseInt(pageSizeStr);
        }
        // make the ListNamedShadowsForThing request
        const response = await this.ipcClient.listNamedShadowsForThing(request);
        const shadowNames = response.results;
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new listNamedShadowsForThing();
```

Administre las implementaciones y los componentes locales

> **Note**
>
> Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass.

Utilice el servicio CLI IPC de Greengrass para administrar las implementaciones locales y los componentes de Greengrass en el dispositivo principal.

Para utilizar estas operaciones de IPC, incluya la versión 2.6.0 o posterior del componente CLI de Greengrass como dependencia en su componente personalizado. A continuación, puede utilizar las operaciones de IPC en sus componentes personalizados para hacer lo siguiente:

- Cree implementaciones locales para modificar y configurar los componentes de Greengrass en el dispositivo principal.
- Reinicie y detenga los componentes de Greengrass en el dispositivo principal.
- Genere una contraseña que pueda usar para iniciar sesión en la consola de depuración local.

Temas

- Versiones mínimas de SDK
- Autorización
- CreateLocalDeployment
- ListLocalDeployments
- GetLocalDeploymentStatus
- ListComponents
- GetComponentDetails
- RestartComponent
- StopComponent
- CreateDebugPassword

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas de las SDK para dispositivos con AWS IoT que debe utilizar para interactuar con el servicio CLI IPC de Greengrass.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.2.10	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.5.3	

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.17.0	
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	

Autorización

Para utilizar el servicio CLI IPC de Greengrass en un componente personalizado, debe definir políticas de autorización que permitan a su componente administrar las implementaciones y los componentes locales. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización de la CLI de Greengrass tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.Cli`

Operación	Description (Descripción)	Recursos
`aws.greengrass#CreateLocalDeployment`	Permite que un componente cree una implementación local en el dispositivo principal.	*
`aws.greengrass#ListLocalDeployments`	Permite que un componente enumere las implementaciones locales en el dispositivo principal.	*
`aws.greengrass#GetLocalDeploymentStatus`	Permite que un componente obtenga el estado de una implementación local en el dispositivo principal.	Un ID de implementación local, o *, que permita el acceso a todas las implementaciones locales.
`aws.greengrass#ListComponents`	Permite que un componente enumere los componentes del dispositivo principal.	*

Operación	Description (Descripción)	Recursos
`aws.greengrass#GetComponentDetails`	Permite que un componente obtenga detalles sobre un componente del dispositivo principal.	Un nombre de componente, por ejemplo: `com.example.HelloWorld` o `*`, que permita el acceso a todos los componentes.
`aws.greengrass#RestartComponent`	Permite que un componente reinicie un componente del dispositivo principal.	Un nombre de componente, por ejemplo: `com.example.HelloWorld` o `*`, que permita el acceso a todos los componentes.
`aws.greengrass#StopComponent`	Permite que un componente detenga a un componente del dispositivo principal.	Un nombre de componente, por ejemplo: `com.example.HelloWorld` o `*`, que permita el acceso a todos los componentes.
`aws.greengrass#CreateDebugPassword`	Permite que un componente genere una contraseña para iniciar sesión en el componente de la consola de depuración local.	`*`

Example Ejemplo de política de autorización

Los siguientes ejemplos de políticas de autorización permiten a un componente crear implementaciones locales, ver todas las implementaciones y componentes locales, además de reiniciar y detener un componente denominado `com.example.HelloWorld`.

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view
 deployments and components.",
```

```
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",
          "aws.greengrass#ListComponents",
          "aws.greengrass#GetComponentDetails"
        ],
        "resources": [
          "*"
        ]
      }
    },
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:2": {
        "policyDescription": "Allows access to restart and stop the Hello World
 component.",
        "operations": [
          "aws.greengrass#RestartComponent",
          "aws.greengrass#StopComponent"
        ],
        "resources": [
          "com.example.HelloWorld"
        ]
      }
    }
  }
}
```

CreateLocalDeployment

Cree o actualice una implementación local mediante recetas de componentes, artefactos y argumentos de tiempo de ejecución específicos.

Esta operación ofrece la misma funcionalidad que el comando crear implementación de la CLI de Greengrass.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`recipeDirectoryPath` (Python: `recipe_directory_path`)

(Opcional) La ruta absoluta a la carpeta que contiene los archivos de recetas de los componentes.

`artifactDirectoryPath` (Python: `artifact_directory_path`)

(Opcional) La ruta absoluta a la carpeta que contiene los archivos de artefactos que se incluirán en la implementación. La carpeta de artefactos debe contener la siguiente estructura de carpetas:

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd` (Python: `root_component_versions_to_add`)

(Opcional) Las versiones de los componentes que se van a instalar en el dispositivo principal. Este objeto, `ComponentToVersionMap`, es un mapa que contiene los siguientes pares clave-valor:

`key`

El nombre del componente.

`value`

Esta es la versión del componente.

`rootComponentsToRemove` (Python: `root_components_to_remove`)

(Opcional) Los componentes que se van a desinstalar del dispositivo principal. Especifique una lista en la que cada entrada sea el nombre de un componente.

`componentToConfiguration` (Python: `component_to_configuration`)

(opcional) La configuración se actualiza para cada componente en la implementación. Este objeto, `ComponentToConfiguration`, es un mapa que contiene los siguientes pares clave-valor:

`key`

El nombre del componente.

`value`

La configuración actualiza el objeto JSON para el componente. El objeto JSON debe incluir el siguiente formato.

```
{
```

```
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

Para obtener más información acerca de las actualizaciones de configuración, consulte Actualización de las configuraciones de los componentes.

`componentToRunWithInfo` (Python: `component_to_run_with_info`)

(Opcional) La configuración del tiempo de ejecución de cada componente de la implementación. Esta configuración incluye el usuario del sistema que es propietario de los procesos de cada componente y los límites del sistema que se aplican a cada componente. Este objeto, `ComponentToRunWithInfo`, es un mapa que contiene los siguientes pares clave-valor:

`key`

El nombre del componente.

`value`

La configuración de tiempo de ejecución del componente. Si omite un parámetro de configuración del tiempo de ejecución, el software AWS IoT Greengrass principal utilizará los valores predeterminados que configure en el núcleo de Greengrass. Este objeto, `RunWithInfo`, contiene la siguiente información:

`posixUser` (Python: `posix_user`)

(Opcional) El usuario del sistema POSIX y, opcionalmente, el grupo que se utilizarán para ejecutarse en dispositivos principales de Linux. El usuario y el grupo, si se especifica, deben existir en cada dispositivo principal de Linux. Especifique el usuario y el grupo separados por dos puntos (`:`) con el siguiente formato: `user:group`. El grupo es opcional. Si no especifica un grupo, el software AWS IoT Greengrass Core utiliza el grupo principal para el usuario. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

`windowsUser` (Python: `windows_user`)

(Opcional) El usuario de Windows que se utilizará para ejecutar este componente en los dispositivos principales de Windows. El usuario debe estar en todos los dispositivos principales de Windows y su nombre y contraseña deben almacenarse en la instancia del

administrador de credenciales de la LocalSystem cuenta. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes.

`systemResourceLimits` (Python: `system_resource_limits`)

(Opcional) Los límites de recursos del sistema que se aplicarán a los procesos de este componente. Puede aplicar límites de recursos del sistema a los componentes de Lambda genéricos y no contenerizados. Para obtener más información, consulte Configuración de los límites de recursos del sistema para los componentes.

AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Este objeto, `SystemResourceLimits`, contiene la siguiente información:

`cpus`

(Opcional) La cantidad máxima de tiempo de CPU que los procesos de un componente pueden utilizar en el dispositivo principal. El tiempo total de CPU de un dispositivo principal equivale a la cantidad de núcleos de CPU del dispositivo. Por ejemplo, en un dispositivo principal con 4 núcleos de CPU, puede establecer este valor en 2 para limitar los procesos del componente al 50 % de uso de cada núcleo de CPU. En un dispositivo con 1 núcleo de CPU, puede establecer este valor en `0.25` para limitar los procesos del componente al 25 % de uso de la CPU. Si establece este valor en un número superior al número de núcleos de la CPU, el software AWS IoT Greengrass Core no limita el uso de la CPU del componente.

`memory`

(Opcional) La cantidad máxima de RAM, expresada en kilobytes, que los procesos de un componente pueden utilizar en el dispositivo principal.

`groupName` (Python: `group_name`)

(Opcional) El nombre del grupo de elementos al que se va a dirigir esta implementación.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`deploymentId` (Python: `deployment_id`)

El ID de la implementación local que creó la solicitud.

ListLocalDeployments

Obtiene el estado de las últimas 10 implementaciones locales.

Esta operación ofrece la misma funcionalidad que el comando enumerar implementación de la CLI de Greengrass.

Solicitud

Esta solicitud de la operación no tiene parámetros.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`localDeployments` (Python: `local_deployments`)

La lista de implementaciones locales. Cada objeto de esta lista es un objeto `LocalDeployment` que contiene la siguiente información:

`deploymentId` (Python: `deployment_id`)

El ID de la implementación local.

`status`

El estado de la implementación local. Esta enumeración, `DeploymentStatus`, tiene los siguientes valores:

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

Obtiene el estado de una implementación local.

Esta operación ofrece la misma funcionalidad que el comando estado de implementación de la CLI de Greengrass.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`deploymentId` (Python: `deployment_id`)

El ID de la implementación local que se va a obtener.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`deployment`

La implementación local. Este objeto, `LocalDeployment`, contiene la siguiente información:

`deploymentId` (Python: `deployment_id`)

El ID de la implementación local.

`status`

El estado de la implementación local. Esta enumeración, `DeploymentStatus`, tiene los siguientes valores:

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

ListComponents

Obtiene el nombre, la versión, el estado y la configuración de cada componente raíz del dispositivo principal. Un componente raíz es un componente que se especifica en una implementación. Esta respuesta no incluye los componentes que se instalan como dependencias de otros componentes.

Esta operación ofrece la misma funcionalidad que el comando component list de la CLI de Greengrass.

Solicitud

Esta solicitud de la operación no tiene parámetros.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`components`

La lista de componentes raíz del dispositivo principal. Cada objeto de esta lista es un objeto `ComponentDetails` que contiene la siguiente información:

`componentName` (Python: `component_name`)

El nombre del componente.

`version`

Esta es la versión del componente.

`state`

El estado del componente. Este estado puede ser uno de los siguientes:

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

La configuración del componente como objeto JSON.

GetComponentDetails

Obtiene la versión, el estado y la configuración de un componente del dispositivo principal.

Esta operación ofrece la misma funcionalidad que el comando detallar implementación de la CLI de Greengrass.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

El nombre del componente que se obtendrá.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`componentDetails` (Python: `component_details`)

Los detalles del componente. Este objeto, `ComponentDetails`, contiene la siguiente información:

`componentName` (Python: `component_name`)

El nombre del componente.

`version`

Esta es la versión del componente.

`state`

El estado del componente. Este estado puede ser uno de los siguientes:

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

La configuración del componente como objeto JSON.

RestartComponent

Reinicia un componente en el dispositivo principal.

> **Note**
>
> Si bien puede reiniciar cualquier componente, le recomendamos que reinicie solo los componentes genéricos.

Esta operación ofrece la misma funcionalidad que el comando reiniciar implementación de la CLI de Greengrass.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

El nombre del componente.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`restartStatus` (Python: `restart_status`)

El estado de la solicitud reiniciada. El estado de la solicitud puede ser uno de los siguientes:

- `SUCCEEDED`
- `FAILED`

`message`

Un mensaje sobre por qué el componente no se pudo reiniciar, si la solicitud falló.

Ejemplos

En los ejemplos siguientes, se muestra cómo llamar a esta operación en código de componente personalizado.

Rust

Example Ejemplo: reiniciar un componente

```
use gg_sdk::Sdk;

fn main() {
    let sdk = Sdk::init();
    sdk.connect().expect("Failed to establish IPC connection");

    let component_name = "com.example.HelloWorld";

    sdk.restart_component(component_name)
        .expect("Failed to restart component");

    println!("Successfully requested restart for component: {component_name}");
}
```

C

Example Ejemplo: reiniciar un componente

```
#include <gg/error.h>
#include <gg/ipc/client.h>
#include <gg/sdk.h>
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    gg_sdk_init();

    GgError err = ggipc_connect();
    if (err != GG_ERR_OK) {
        fprintf(stderr, "Failed to establish IPC connection.\n");
        exit(-1);
    }

    GgBuffer component_name = GG_STR("com.example.HelloWorld");

    err = ggipc_restart_component(component_name);
    if (err != GG_ERR_OK) {
        fprintf(
```

```
            stderr,
            "Failed to restart component: %.*s\n",
            (int) component_name.len,
            component_name.data
        );
        exit(-1);
    }

    printf(
        "Successfully requested restart for component: %.*s\n",
        (int) component_name.len,
        component_name.data
    );
}
```

C++ (Component SDK)

Example Ejemplo: reiniciar un componente

```
#include <gg/ipc/client.hpp>
#include <iostream>

int main() {
    auto &client = gg::ipc::Client::get();

    auto error = client.connect();
    if (error) {
        std::cerr << "Failed to establish IPC connection.\n";
        exit(-1);
    }

    std::string_view component_name = "com.example.HelloWorld";

    error = client.restart_component(component_name);
    if (error) {
        std::cerr << "Failed to restart component: " << component_name << "\n";
        exit(-1);
    }

    std::cout << "Successfully requested restart for component: "
              << component_name << "\n";
}
```

StopComponent

Detiene los procesos de un componente en el dispositivo principal.

> **Note**
>
> Si bien puede detener cualquier componente, le recomendamos que detenga solo los componentes genéricos.

Esta operación ofrece la misma funcionalidad que el comando detener implementación de la CLI de Greengrass.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`componentName` (Python: `component_name`)

El nombre del componente.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`stopStatus` (Python: `stop_status`)

El estado de la solicitud de detención. El estado de la solicitud puede ser uno de los siguientes:

- `SUCCEEDED`
- `FAILED`

`message`

Un mensaje sobre por qué el componente no se pudo detener, si la solicitud falló.

CreateDebugPassword

Genera una contraseña asignada al azar que puede usar para iniciar sesión en el componente de la consola de depuración local. La contraseña caduca 8 horas después de generarse.

Esta operación proporciona la misma funcionalidad que el get-debug-password comando de la CLI de Greengrass.

Solicitud

Esta solicitud de la operación no tiene parámetros.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`username`

El nombre de usuario que se utilizará para iniciar sesión.

`password`

La contraseña que se utilizará para iniciar sesión.

`passwordExpiration` (Python: `password_expiration`)

La hora en que caduca la contraseña.

`certificateSHA256Hash` (Python: `certificate_sha256_hash`)

La huella digital SHA-256 del certificado autofirmado que utiliza la consola de depuración local cuando HTTPS está activado. Cuando abra la consola de depuración local, utilice esta huella digital para comprobar que el certificado es legítimo y que la conexión es segura.

`certificateSHA1Hash` (Python: `certificate_sha1_hash`)

La huella digital SHA-1 del certificado autofirmado que utiliza la consola de depuración local cuando HTTPS está activado. Cuando abra la consola de depuración local, utilice esta huella digital para comprobar que el certificado es legítimo y que la conexión es segura.

Autenticación y autorización de los dispositivos de cliente

> **Note**
>
> Esta característica está disponible para la versión 2.6.0 y versiones posteriores del componente núcleo de Greengrass.

Utilice el servicio IPC de autenticación de dispositivos de cliente para desarrollar un componente de agente local personalizado al que puedan conectarse los dispositivos IoT locales, como los dispositivos de cliente.

Para utilizar estas operaciones de IPC, incluya la versión 2.2.0 o posterior del componente de autorización del dispositivo de cliente como dependencia en su componente personalizado. A continuación, puede utilizar las operaciones de IPC en sus componentes personalizados para hacer lo siguiente:

- Compruebe la identidad de los dispositivos de cliente que se conectan al dispositivo principal.
- Cree una sesión para que un dispositivo de cliente se conecte al dispositivo principal.
- Compruebe si un dispositivo de cliente tiene permiso para realizar una acción.
- Reciba una notificación cuando el certificado del servidor del dispositivo principal cambie.

Temas

- Versiones mínimas de SDK
- Autorización
- VerifyClientDeviceIdentity
- GetClientDeviceAuthToken
- AuthorizeClientDeviceAction
- SubscribeToCertificateUpdates

Versiones mínimas de SDK

En la siguiente tabla se enumeran las versiones mínimas del SDK para dispositivos con AWS IoT que debe utilizar para interactuar con el servicio IPC de autenticación del dispositivo cliente.

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para Java v2	Versión 1.9.3	
SDK para dispositivos con AWS IoT para Python v2	Versión 1.11.3	

SDK	Versión mínima	
SDK para dispositivos con AWS IoT para C++ v2	Versión 1.18.3	
SDK para dispositivos con AWS IoT para JavaScript v2	Versión 1.12.0	

Autorización

Para utilizar el servicio IPC de autenticación del dispositivo de cliente en un componente personalizado, debe definir políticas de autorización que permitan a su componente realizar estas operaciones. Para obtener información sobre cómo definir las políticas de autorización, consulte Autorización de los componentes para realizar operaciones de IPC.

Las políticas de autorización para la autenticación y autorización de los dispositivos de cliente tienen las siguientes propiedades.

Identificador de servicio IPC: `aws.greengrass.clientdevices.Auth`

Operación	Description (Descripción)	Recursos
`aws.greengrass#VerifyClientDeviceIdentity`	Permite que un componente verifique la identidad de un dispositivo de cliente.	*
`aws.greengrass#GetClientDeviceAuthToken`	Permite que un componente valide las credenciales de un dispositivo de cliente y cree una sesión para ese dispositivo de cliente.	*
`aws.greengrass#AuthorizeClientDeviceAction`	Permite que un componente verifique si un dispositivo de cliente tiene permiso para realizar una acción.	*

Operación	Description (Descripción)	Recursos
`aws.greengrass#SubscribeToCertificateUpdates`	Permite que un componente reciba notificaciones cuando el certificado del servidor del dispositivo principal rota.	*
*	Permite que un componente realice todas las operaciones del servicio IPC de autenticación del dispositivo de cliente.	*

Ejemplos de políticas de autorización

Puede consultar el siguiente ejemplo de política de autorización con el fin de configurar las políticas de autorización para sus componentes.

Example Ejemplo de política de autorización

El siguiente ejemplo de política de autorización permite a un componente realizar todas las operaciones de IPC de autenticación del dispositivo de cliente.

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
 devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
```

```
}
```

VerifyClientDeviceIdentity

Compruebe la identidad de un dispositivo de cliente. Esta operación verifica si el dispositivo cliente es válido AWS IoT .

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`credential`

Las credenciales del dispositivo de cliente. Este objeto, `ClientDeviceCredential`, contiene la siguiente información:

`clientDeviceCertificate` (Python: `client_device_certificate`)

El certificado de dispositivo X.509 del dispositivo de cliente.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`isValidClientDevice` (Python: `is_valid_client_device`)

Si la identidad del dispositivo de cliente es válida.

GetClientDeviceAuthToken

Valida las credenciales de un dispositivo de cliente y crea una sesión para el dispositivo de cliente. Esta operación devuelve un token de sesión que puede usar en solicitudes posteriores para autorizar acciones del dispositivo de cliente.

Para conectar correctamente un dispositivo de cliente, el componente de autenticación del dispositivo de cliente debe conceder el permiso `mqtt:connect` para el ID de cliente que utiliza el dispositivo de cliente.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`credential`

Las credenciales del dispositivo de cliente. Este objeto, `CredentialDocument`, contiene la siguiente información:

`mqttCredential` (Python: `mqtt_credential`)

Las credenciales MQTT del dispositivo de cliente. Especifique el ID de cliente y el certificado que el dispositivo de cliente utiliza para conectarse. Este objeto, `MQTTCredential`, contiene la siguiente información:

`clientId` (Python: `client_id`)

El ID de cliente que se utilizará para conectarse.

`certificatePem` (Python: `certificate_pem`)

El certificado del dispositivo X.509 que se utilizará para conectarse.

`username`

> **Note**
>
> En la actualidad, esta propiedad no se utiliza.

`password`

> **Note**
>
> En la actualidad, esta propiedad no se utiliza.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`clientDeviceAuthToken` (Python: `client_device_auth_token`)

El token de sesión del dispositivo de cliente. Puede usar este token de sesión en solicitudes posteriores para autorizar las acciones de este dispositivo de cliente.

AuthorizeClientDeviceAction

Compruebe si un dispositivo de cliente tiene permiso para realizar una acción en un recurso. Las políticas de autorización de los dispositivos de cliente especifican los permisos que los dispositivos de cliente pueden realizar mientras están conectados a un dispositivo principal. Las políticas de autorización de los dispositivos de cliente se definen al configurar el componente de autenticación del dispositivo de cliente.

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`clientDeviceAuthToken` (Python: `client_device_auth_token`)

El token de sesión del dispositivo de cliente.

`operation`

La operación que se va a autorizar.

`resource`

El recurso en el que el dispositivo de cliente realiza la operación.

Respuesta

Esta respuesta de operación contiene la siguiente información:

`isAuthorized` (Python: `is_authorized`)

Si el dispositivo de cliente está autorizado a realizar la operación en el recurso.

SubscribeToCertificateUpdates

Suscríbase para recibir el nuevo certificado de servidor del dispositivo principal cada vez que vaya rotando. Cuando el certificado de servidor rota, los agentes deben volver a cargarlo con el nuevo certificado de servidor.

De forma predeterminada, el componente de autenticación del dispositivo de cliente rota los certificados de servidor cada 7 días. Puede configurar el intervalo de rotación entre 2 y 10 días.

Esta es una operación de suscripción en la que se suscribe a un flujo de mensajes de eventos. Para usar esta operación, defina un identificador de respuesta de flujo con funciones que gestionen los mensajes de eventos, los errores y el cierre del flujo. Para obtener más información, consulte Suscripción a los flujos de eventos de IPC.

Tipo de mensaje del evento: `CertificateUpdateEvent`

Solicitud

Esta solicitud de operación tiene los siguientes parámetros:

`certificateOptions` (Python: `certificate_options`)

Los tipos de actualizaciones de certificados a las que debe suscribirse. Este objeto, `CertificateOptions`, contiene la siguiente información:

`certificateType` (Python: `certificate_type`)

El tipo de actualizaciones de certificado a las que se debe suscribir. Elija la opción siguiente:

- `SERVER`

Respuesta

Esta respuesta de operación contiene la siguiente información:

`messages`

El flujo de mensajes. Este objeto, `CertificateUpdateEvent`, contiene la siguiente información:

`certificateUpdate` (Python: `certificate_update`)

La información acerca del nuevo certificado. Este objeto, `CertificateUpdate`, contiene la siguiente información:

`certificate`

El certificado.

`privateKey` (Python: `private_key`)

La clave privada del certificado.

`publicKey` (Python: `public_key`)

La clave pública del certificado.

`caCertificates` (Python: `ca_certificates`)

La lista de los certificados de la autoridad de certificación (CA) de la cadena de certificados de la CA del certificado.

Interacción con dispositivos IoT locales

Los dispositivos de cliente son dispositivos IoT locales que se conectan y se comunican con un dispositivo principal de Greengrass a través de MQTT. Para permitir que los dispositivos de cliente se conecten a un dispositivo principal, haga lo siguiente:

- Interactuar con los mensajes MQTT en los componentes de Greengrass.
- Reenviar mensajes y datos entre los dispositivos de cliente y AWS IoT Core.
- Interactuar con las sombras de dispositivo de cliente en los componentes de Greengrass.
- Sincronizar las sombras de dispositivo de cliente con AWS IoT Core.
- Usar IPv6 para la mensajería local.

Para conectarse a un dispositivo principal, los dispositivos de cliente pueden usar la detección en la nube. Los dispositivos de cliente se conectan al servicio en la nube de AWS IoT Greengrass para recuperar información sobre los dispositivos principales a los que se pueden conectar. Luego, pueden conectarse a un dispositivo principal para procesar sus mensajes y sincronizar sus datos con el servicio en la nube de AWS IoT Core.

Puede seguir un tutorial que lo guiará cómo configurar un dispositivo principal para conectarse y comunicarse con cualquier objeto AWS IoT. En este tutorial, también se explica cómo desarrollar un componente personalizado de Greengrass que interactúe con los dispositivos de cliente. Para obtener más información, consulte Tutorial: interactúe con dispositivos IoT locales a través de MQTT.

Temas

- Componentes de dispositivo de cliente proporcionados por AWS
- Conexión de dispositivos de cliente a los dispositivos principales
- Retransmitir mensajes MQTT entre dispositivos cliente y AWS IoT Core
- Interacción con los dispositivos de cliente en los componentes
- Interacción y sincronización con las sombras de dispositivo de cliente
- IPv6 Úselo para mensajería local
- Solución de problemas de los dispositivos de cliente

Componentes de dispositivo de cliente proporcionados por AWS

AWS IoT Greengrass ofrece los siguientes componentes públicos que puede implementar en los dispositivos principales. Estos componentes permiten que los dispositivos de cliente se conecten y se comuniquen con un dispositivo principal.

> **Note**
>
> Varios de los componentes proporcionados por AWS dependen de versiones secundarias específicas del núcleo de Greengrass. Debido a esta dependencia, es necesario actualizar estos componentes al actualizar el núcleo de Greengrass a una nueva versión secundaria. Para obtener información sobre las versiones específicas del núcleo de las que depende cada componente, consulte el tema del componente correspondiente. Para más información sobre la actualización del núcleo, consulte Actualización del software AWS IoT Greengrass Core (OTA).

Cuando un componente tiene un tipo de componente genérico y de Lambda, la versión actual del componente es del tipo genérico y la versión anterior del componente es del tipo de Lambda.

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto
Autenticación del dispositivo de cliente	Habilita los dispositivos IoT locales, denominados dispositivos de cliente, a conectarse al dispositivo principal.	Complemento	Linux, Windows	Sí
Detector de IP	Envía la información de conectivi	Complemento	Linux, Windows	Sí

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto
	dad del agente MQTT a AWS IoT Greengrass, para que los dispositivos de cliente puedan descubrir cómo conectarse.			
Puente MQTT	Retransmite mensajes MQTT entre los dispositivos cliente, los de publicación/suscripción local de AWS IoT Greengrass e AWS IoT Core.	Complemento	Linux, Windows	Sí
Agente MQTT 3.1.1 (Moquette)	Ejecuta un agente MQTT 3.1.1 que administra los mensajes entre los dispositivos de cliente y el dispositivo principal.	Complemento	Linux, Windows	Sí

Componente	Descripción	Tipo de componente	Sistema operativo admitido	Código abierto
Agente MQTT 5 (EMQX)	Ejecuta un agente MQTT 5 que administra los mensajes entre los dispositivos de cliente y el dispositivo principal.	Genérico	Linux, Windows	No
Administrador de sombras	Permite la interacción con las sombras del dispositivo principal. Administra el almacenamiento de documentos de sombra y también la sincronización de los estados de sombras locales con el servicio de sombra de dispositivo de AWS IoT.	Complemento	Linux, Windows	Sí

Conexión de dispositivos de cliente a los dispositivos principales

Puede configurar la detección en la nube para conectar los dispositivos de cliente a los dispositivos principales. Al configurar la detección en la nube, los dispositivos cliente se pueden conectar al servicio AWS IoT Greengrass en la nube para recuperar información sobre los dispositivos principales a los que se pueden conectar. A continuación, los dispositivos de cliente pueden intentar conectarse a cada dispositivo principal hasta que se conecten correctamente.

Para usar la detección en la nube, debe hacer lo siguiente:

- Asocie los dispositivos de cliente a los dispositivos principales a los que se pueden conectar.
- Especifique los puntos de conexión del agente MQTT donde los dispositivos de cliente se pueden conectar a cada dispositivo principal.
- Implemente componentes en el dispositivo principal que permitan la compatibilidad con los dispositivos de cliente.

  Puede implementar componentes opcionales para hacer lo siguiente:

  - Retransmita mensajes entre los dispositivos cliente, los componentes de Greengrass y el servicio AWS IoT Core en la nube.
  - Administre automáticamente los puntos de conexión del agente MQTT de los dispositivos principales para usted.
  - Gestione las sombras de los dispositivos cliente locales y sincronícelas con el servicio AWS IoT Core en la nube.

También debe revisar y actualizar la AWS IoT política del dispositivo principal para comprobar que tiene los permisos necesarios para conectar los dispositivos cliente. Para obtener más información, consulte Requisitos.

Después de configurar la detección en la nube, puede probar las comunicaciones entre un dispositivo de cliente y un dispositivo principal. Para obtener más información, consulte Prueba de las comunicaciones del dispositivo de cliente.

Temas

- Requisitos
- Componentes de Greengrass para compatibilidad con dispositivos de cliente
- Configuración de la detección en la nube (consola)

- Configuración de la detección en la nube (AWS CLI)
- Asociación de los dispositivos de cliente
- Autenticación de clientes sin conexión
- Administración de puntos de conexión del dispositivo principal
- Elección de un agente MQTT
- Conexión de dispositivos cliente a un dispositivo AWS IoT Greengrass Core con un intermediario MQTT
- Prueba de las comunicaciones del dispositivo de cliente
- API de descubrimiento de Greengrass RESTful

Requisitos

Para conectar dispositivos de cliente a un dispositivo principal, debe contar con lo siguiente:

- El dispositivo principal debe ejecutar la versión 2.2.0 o posterior del núcleo de Greengrass.
- La función de servicio de Greengrass asociada a usted AWS IoT Greengrass Cuenta de AWS en la AWS región en la que opera el dispositivo principal. Para obtener más información, consulte Configuración del rol de servicio de Greengrass.
- La AWS IoT política del dispositivo principal debe permitir los siguientes permisos:
  - `greengrass:PutCertificateAuthorities`
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
  - `greengrass:GetConnectivityInfo`
  - `greengrass:UpdateConnectivityInfo`— (Opcional) Este permiso es necesario para utilizar el componente detector de IP, que envía la información de conectividad de red del dispositivo principal al servicio AWS IoT Greengrass en la nube.
  - `iot:GetThingShadow`iot:UpdateThingShadow, y `iot:DeleteThingShadow` — (opcional) Estos permisos son necesarios para utilizar el componente administrador de sombras con el que se sincronizan las sombras de los dispositivos cliente AWS IoT Core. Esta característica requiere la versión 2.6.0 o versiones posteriores del núcleo de Greengrass, la versión 2.2.0 y versiones posteriores del administrador de sombras y la versión 2.2.0 o versiones posteriores del puente de MQTT.

  Para obtener más información, consulte Configure la política de AWS IoT cosas.

> **Note**
>
> Si utilizó la AWS IoT política predeterminada al instalar el software AWS IoT Greengrass Core, el dispositivo principal tiene una AWS IoT política que permite el acceso a todas AWS IoT Greengrass las acciones (`greengrass:*`).

- AWS IoT cosas que se pueden conectar como dispositivos cliente. Para obtener más información, consulte Crear recursos de AWS IoT en la Guía para desarrolladores de AWS IoT Core .
- Un dispositivo de cliente debe conectarse mediante un ID de cliente. Un ID de cliente es el nombre de un objeto. No se aceptará ningún otro ID de cliente.
- La AWS IoT política de cada dispositivo cliente debe permitir el `greengrass:Discover` permiso. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos cliente.

Temas

- Configuración del rol de servicio de Greengrass
- Configure la política de AWS IoT cosas

Configuración del rol de servicio de Greengrass

La función de servicio de Greengrass es una función de servicio AWS Identity and Access Management (IAM) que autoriza el acceso AWS IoT Greengrass a los recursos de los AWS servicios en su nombre. Esta función permite verificar la identidad de AWS IoT Greengrass los dispositivos cliente y administrar la información de conectividad principal de los dispositivos.

Si no ha configurado anteriormente la función de servicio de Greengrass en esta región, debe asociarle una función de servicio de Greengrass en esta región. AWS IoT Greengrass Cuenta de AWS

Cuando utiliza la página Configurar la detección de dispositivos principales de la AWS IoT Greengrass consola, AWS IoT Greengrass configura el rol de servicio Greengrass automáticamente. De lo contrario, puede configurarlo manualmente mediante la AWS IoT consola o la AWS IoT Greengrass API.

En esta sección, comprueba si el rol de servicio de Greengrass está configurado. Si no está configurado, crea un nuevo rol de servicio de Greengrass AWS IoT Greengrass para asociarlo con usted Cuenta de AWS en esta región.

Configuración del rol de servicio de Greengrass (consola)

1. Compruebe si la función de servicio de Greengrass está asociada AWS IoT Greengrass a su función Cuenta de AWS en esta región. Haga lo siguiente:

   a. Vaya a la consola de AWS IoT.

   b. En el panel de navegación, seleccione Configuración.

   c. En la sección Rol de servicio de Greengrass, encuentre Rol de servicio actual para ver si hay un rol de servicio de Greengrass asociado.

      Si tiene un rol de servicio de Greengrass asociado, cumple con este requisito para usar el componente detector de IP. Vaya a Configure la política de AWS IoT cosas.

2. Si el rol de servicio de Greengrass no está asociado AWS IoT Greengrass para usted Cuenta de AWS en esta región, cree un rol de servicio de Greengrass y asócielo. Haga lo siguiente:

   a. Vaya a la consola de IAM.

   b. Elija Roles.

   c. Elija Crear rol.

   d. En la página Crear rol, haga lo siguiente:

      i. En Tipo de entidad de confianza, elija Servicio de AWS.

      ii. En Caso de uso, Casos de uso para otros Servicios de AWS, elija Greengrass y seleccione Greengrass. Esta opción especifica agregarlo AWS IoT Greengrass como entidad de confianza que pueda asumir este rol.

      iii. Elija Siguiente.

      iv. En Políticas de permisos, elija la AWSGreengrassResourceAccessRolePolicy que desee adjuntar al rol.

      v. Elija Siguiente.

      vi. En Nombre del rol, ingrese un nombre para su rol, por ejemplo, **`Greengrass_ServiceRole`**.

      vii. Elija Crear rol.

   e. Vaya a la consola de AWS IoT.

   f. En el panel de navegación, seleccione Configuración.

   g. En la sección rol de servicio de Greengrass, elija Adjuntar rol.

h. En el cuadro Actualizar rol de servicio de Greengrass, elija el rol de IAM que creó y, a continuación, elija Adjuntar rol.

Configuración del rol de servicio de Greengrass (AWS CLI)

1. Compruebe si la función de servicio de Greengrass está asociada AWS IoT Greengrass a su función Cuenta de AWS en esta región.

```
aws greengrassv2 get-service-role-for-account
```

Si el rol de servicio de Greengrass está asociado, la operación devuelve una respuesta que contiene información sobre el rol.

Si tiene un rol de servicio de Greengrass asociado, cumple con este requisito para usar el componente detector de IP. Vaya a Configure la política de AWS IoT cosas.

2. Si el rol de servicio de Greengrass no está asociado AWS IoT Greengrass para usted Cuenta de AWS en esta región, cree un rol de servicio de Greengrass y asócielo. Haga lo siguiente:

a. Cree un rol con una política de confianza que le permita AWS IoT Greengrass asumir el rol. Este ejemplo crea un rol denominado `Greengrass_ServiceRole`, pero puede utilizar un nombre distinto. Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención de la sustitución confusa entre servicios.

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document "{\\"Version\\":\\"2012-10-17      \\",\\"Statement\\":[{\\"Effect
\\":\\"Allow\\",\\"Principal\\":{\\"Service\\":\\"greengrass.amazonaws.com
\\"},\\"Action\\":\\"sts:AssumeRole\\",\\"Condition\\":{\\"ArnLike\\":
{\\"aws:SourceArn\\":\\"arn:aws:greengrass:region:account-id:*\\"},\
\\"StringEquals\\":{\\"aws:SourceAccount\\":\\"account-id\\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
```

```
  ]
}'
```

b. Copie el ARN del rol de los metadatos del rol en la salida. Puede utilizar el ARN para asociar el rol a su cuenta.

c. Asocie la política de `AWSGreengrassResourceAccessRolePolicy` al rol.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

d. Asocie la función de servicio de Greengrass con AWS IoT Greengrass su. Cuenta de AWS*role-arn*Sustitúyalo por el ARN de la función de servicio.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

La operación devuelve la siguiente respuesta si tiene éxito.

```
{
  "associatedAt": "timestamp"
}
```

Configure la política de AWS IoT cosas

Los dispositivos principales usan certificados de dispositivo X.509 para autorizar las conexiones a AWS. Debe adjuntar políticas AWS IoT a los certificados de los dispositivos para definir los permisos de un dispositivo principal. Para obtener más información, consulte AWS IoT políticas para las operaciones del plano de datos y AWS IoT Política mínima de compatibilidad con los dispositivos cliente.

Para conectar los dispositivos cliente a un dispositivo principal, la AWS IoT política del dispositivo principal debe permitir los siguientes permisos:

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`

- `greengrass:UpdateConnectivityInfo`— (Opcional) Este permiso es necesario para usar el componente detector de IP, que envía la información de conectividad de red del dispositivo principal al servicio AWS IoT Greengrass en la nube.
- `iot:GetThingShadow`, `iot:UpdateThingShadow`, y `iot:DeleteThingShadow` — (opcional) Estos permisos son necesarios para utilizar el componente administrador de sombras con el que se sincronizan las sombras de los dispositivos cliente AWS IoT Core. Esta característica requiere la versión 2.6.0 o versiones posteriores del núcleo de Greengrass, la versión 2.2.0 y versiones posteriores del administrador de sombras y la versión 2.2.0 o versiones posteriores del puente de MQTT.

En esta sección, revisará las AWS IoT políticas de su dispositivo principal y agregará los permisos necesarios que falten. Si utilizaste el instalador del software AWS IoT Greengrass principal para aprovisionar recursos, tu dispositivo principal tiene una AWS IoT política que permite el acceso a todas AWS IoT Greengrass las acciones (`greengrass:*`). En este caso, solo debe actualizar la AWS IoT política si planea implementar el componente de administrador de sombras para sincronizar las sombras de los dispositivos con él AWS IoT Core. De lo contrario, puede omitir esta sección.

Configure la política de AWS IoT cosas (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Dispositivos principales.
2. En la página Dispositivos principales, elija el dispositivo principal que desea actualizar.
3. En la página de detalles del dispositivo principal, elija el enlace al Objeto del dispositivo principal. Este enlace abre la página de detalles del objeto en la consola de AWS IoT .
4. En la página de detalles del objeto, elija Certificados.
5. En la pestaña Certificados, elija el certificado activo del objeto.
6. En la página de detalles del certificado, elija Políticas.
7. En la pestaña Políticas, elija la AWS IoT política que desee revisar y actualizar. Puede agregar los permisos necesarios a cualquier política que esté asociada al certificado activo del dispositivo principal.

> **Note**
>
> Si ha utilizado el instalador de software AWS IoT Greengrass principal para aprovisionar recursos, tiene dos AWS IoT políticas. Le recomendamos que elija la política denominada GreengrassV2IoTThingPolicy, si existe. Los dispositivos principales que cree con el instalador rápido usan este nombre de política de forma predeterminada. Si

agrega permisos a esta política, también los otorga a otros dispositivos principales que usan esta política.

8. En la descripción general de la política, elija Editar la versión activa.
9. Revise la política para ver los permisos necesarios y agregue los permisos necesarios que falten.

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`— (Opcional) Este permiso es necesario para utilizar el componente de detección de IP, que envía la información de conectividad de red del dispositivo principal al servicio AWS IoT Greengrass en la nube.
   - `iot:GetThingShadow`iot:UpdateThingShadow, y `iot:DeleteThingShadow` — (opcional) Estos permisos son necesarios para utilizar el componente administrador de sombras con el que se sincronizan las sombras de los dispositivos cliente AWS IoT Core. Esta característica requiere la versión 2.6.0 o versiones posteriores del núcleo de Greengrass, la versión 2.2.0 y versiones posteriores del administrador de sombras y la versión 2.2.0 o versiones posteriores del puente de MQTT.
10. (Opcional) Para permitir que el dispositivo principal se sincronice con las sombras AWS IoT Core, añada la siguiente declaración a la política. Si planea interactuar con las sombras de los dispositivos cliente, pero no sincronizarlas con ellas AWS IoT Core, omita este paso. Sustituya *`region`* y *`account-id`* por la región que utilice y su Cuenta de AWS número.

    - Esta instrucción de ejemplo permite acceder a las sombras de dispositivos de todos los objetos. Para seguir las prácticas recomendadas de seguridad, puede restringir el acceso únicamente al dispositivo principal y a los dispositivos de cliente que conecte al dispositivo principal. Para obtener más información, consulte AWS IoT Política mínima de compatibilidad con los dispositivos cliente.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
```

```
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

Después de agregar esta instrucción, el documento de política tendrá un aspecto semejante al del siguiente ejemplo.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/*"
      ]
    }
  ]
}
```

11. Para establecer una nueva versión de la política como la versión activa, en Estado de la versión de la política, seleccione Establecer la versión editada como la versión activa de esta política.
12. Seleccione Guardar como versión nueva.

Configura la política de AWS IoT cosas (AWS CLI)

1. Enumere los principios del AWS IoT dispositivo principal. Las entidades principales del objeto pueden ser certificados de dispositivo X.509 u otros identificadores. Ejecute el siguiente comando y *MyGreengrassCore* sustitúyalo por el nombre del dispositivo principal.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   La operación devuelve una respuesta que enumera las entidades principales del objeto del dispositivo principal.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. Identifique el certificado activo del dispositivo principal. Ejecute el siguiente comando y *certificateId* sustitúyalo por el ID de cada certificado del paso anterior hasta que encuentre el certificado activo. El ID del certificado es la cadena hexadecimal que se encuentra al final del ARN del certificado. El argumento `--query` especifica que solo se muestre el estado del certificado.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   La operación devuelve el estado del certificado en forma de cadena. Por ejemplo, si el certificado está activo, la operación muestra `"ACTIVE"`.

3. Enumere las AWS IoT políticas que se adjuntan al certificado. Ejecute el siguiente comando y reemplace ARN del certificado por el ARN del certificado.

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

La operación devuelve una respuesta en la que se enumeran AWS IoT las políticas adjuntas al certificado.

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. Elija la política que desee ver y actualizar.

> **Note**
>
> Si utilizó el instalador de software AWS IoT Greengrass principal para aprovisionar recursos, tiene dos AWS IoT políticas. Le recomendamos que elija la política denominada GreengrassV2IoTThingPolicy, si existe. Los dispositivos principales que cree con el instalador rápido usan este nombre de política de forma predeterminada. Si agrega permisos a esta política, también los otorga a otros dispositivos principales que usan esta política.

5. Obtenga el documento de la política. Ejecute el siguiente comando y *GreengrassV2IoTThingPolicy* sustitúyalo por el nombre de la política.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

La operación devuelve una respuesta que contiene el documento de política y otra información sobre la política. El documento de política es un objeto JSON serializado como una cadena.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17       \\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Use un conversor en línea u otra herramienta para convertir la cadena del documento de la política en un objeto JSON y, a continuación, guárdela en un archivo denominado `iot-policy.json`.

   Por ejemplo, si tiene instalada la herramienta jq, puede ejecutar el siguiente comando para obtener el documento de la política, convertirlo en un objeto JSON y guardar el documento de la política como un objeto JSON.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. Revise la política para ver los permisos necesarios y agregue los permisos necesarios que falten.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para abrir el archivo.

   ```
   nano iot-policy.json
   ```

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`— (Opcional) Este permiso es necesario para usar el componente detector de IP, que envía la información de conectividad de red del dispositivo principal al servicio AWS IoT Greengrass en la nube.
   - `iot:GetThingShadow`, `iot:UpdateThingShadow`, y `iot:DeleteThingShadow` — (opcional) Estos permisos son necesarios para utilizar el componente administrador de sombras con el que se sincronizan las sombras de los dispositivos cliente AWS IoT Core. Esta característica requiere la versión 2.6.0 o versiones posteriores del núcleo de Greengrass, la versión 2.2.0 y versiones posteriores del administrador de sombras y la versión 2.2.0 o versiones posteriores del puente de MQTT.

8. Guarde los cambios como una nueva versión de la política. Ejecute el siguiente comando y *`GreengrassV2IoTThingPolicy`* sustitúyalo por el nombre de la política.

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   Si se realiza correctamente, la operación devuelve una respuesta similar a la del siguiente ejemplo.

   ```
   {
       "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy",
       "policyDocument": "{\
     \\"Version\\": \\"2012-10-17       \\",\
     \\"Statement\\": [\
       {\
   ```

```
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

Componentes de Greengrass para compatibilidad con dispositivos de cliente

> **Important**
>
> El dispositivo principal debe ejecutar la versión 2.2.0 o posterior del núcleo de Greengrass para que sea compatible con los dispositivos de cliente.

Para permitir que los dispositivos de cliente se conecten y se comuniquen con un dispositivo principal, debe implementar los siguientes componentes de Greengrass en el dispositivo principal:

- Autenticación del dispositivo de cliente (`aws.greengrass.clientdevices.Auth`)

  Implemente el componente de autenticación del dispositivo de cliente para autenticar los dispositivos de cliente y autorizar las acciones de los dispositivos de cliente. Este componente permite AWS IoT que tus cosas se conecten a un dispositivo principal.

  Este componente requiere alguna configuración para poder usarlo. Debe especificar los grupos de dispositivos de cliente y las operaciones que cada grupo está autorizado a realizar, como conectarse y comunicarse a través de MQTT. Para obtener más información, consulte configuración del componente de autenticación del dispositivo de cliente.
- Agente MQTT 3.1.1 (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

Implemente el componente agente MQTT de Moquette para ejecutar un agente MQTT ligero. El agente MQTT de Moquette es compatible con MQTT 3.1.1 e incluye compatibilidad local para QoS 0, QoS 1, QoS 2, mensajes retenidos, mensajes de última voluntad y suscripciones persistentes.

No es necesario configurar este componente para usarlo. Sin embargo, puede configurar el puerto donde este componente hace funcionar el agente MQTT. Usa el puerto 8883 de manera predeterminada.

- Agente MQTT 5 (EMQX) (`aws.greengrass.clientdevices.mqtt.EMQX`)

  > **Note**
  >
  > Para usar el agente MQTT 5 de EMQX, debe usar la versión 2.6.0 o posterior del núcleo de Greengrass y la versión 2.2.0 o posterior de la autenticación del dispositivo de cliente.

  Implemente el componente agente MQTT de EMQX para usar las características de MQTT 5.0 en la comunicación entre los dispositivos de cliente y el dispositivo principal. El agente MQTT de EMQX es compatible con MQTT 5.0 e incluye compatibilidad para los intervalos de caducidad de las sesiones y los mensajes, las propiedades de los usuarios, las suscripciones compartidas, los alias de los temas y más.

  No es necesario configurar este componente para usarlo. Sin embargo, puede configurar el puerto donde este componente hace funcionar el agente MQTT. Usa el puerto 8883 de manera predeterminada.

- Puente MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Opcional) Implemente el componente puente MQTT para retransmitir mensajes entre los dispositivos cliente (MQTT local), la publicación/suscripción local y el MQTT. AWS IoT Core Configure este componente para sincronizar los dispositivos cliente AWS IoT Core e interactuar con los dispositivos cliente de los componentes de Greengrass.

  Para poder usar este componente, es necesario configurarlo. Debe especificar las asignaciones de temas en las que este componente retransmite los mensajes. Para obtener más información, consulte la configuración del componente puente de MQTT.

- Detector de IP (`aws.greengrass.clientdevices.IPDetector`)

(Opcional) Implemente el componente detector de IP para informar automáticamente al servicio en la nube de los puntos finales del agente MQTT del AWS IoT Greengrass dispositivo principal. No puede usar este componente si tiene una configuración de red compleja, como una en la que un enrutador reenvía el puerto agente MQTT al dispositivo principal.

No es necesario configurar este componente para usarlo.

- Administrador de sombras (`aws.greengrass.ShadowManager`)

> **Note**
>
> Para administrar las sombras de dispositivos de cliente, debe usar la versión 2.6.0 o posterior del núcleo de Greengrass, la versión 2.2.0 o posterior del administrador de sombras y la versión 2.2.0 o posterior del puente de MQTT.

(Opcional) Implemente el componente administrador de sombras para administrar las sombras de dispositivos de cliente en el dispositivo principal. Los componentes de Greengrass pueden obtener, actualizar y eliminar las sombras de dispositivos de cliente para interactuar con los dispositivos de cliente. También puede configurar el componente administrador de sombras para sincronizar las sombras de los dispositivos cliente con el servicio en la AWS IoT Core nube.

Para usar este componente con las sombras de dispositivos de cliente, debe configurar el componente puente de MQTT para que retransmita los mensajes entre los dispositivos de cliente y el administrador de sombras, que usa la función de publicación/suscripción local. De lo contrario, no es necesario configurar este componente para su uso, pero sí para sincronizar las sombras de dispositivos.

> **Note**
>
> Se recomienda implementar solo un componente agente MQTT. Los componentes puente de MQTT y detector de IP solo funcionan con un componente agente MQTT a la vez. Si implementa varios componentes agente de MQTT, debe configurarlos para que usen puertos diferentes.

Configuración de la detección en la nube (consola)

Puede usar la AWS IoT Greengrass consola para asociar los dispositivos cliente, administrar los puntos finales de los dispositivos principales e implementar componentes para habilitar la compatibilidad con los dispositivos cliente. Para obtener más información, consulte Paso 2: Habilitar la compatibilidad del dispositivo de cliente.

Configuración de la detección en la nube (AWS CLI)

Puede usar AWS Command Line Interface (AWS CLI) para asociar los dispositivos cliente, administrar los puntos finales de los dispositivos principales e implementar componentes para habilitar la compatibilidad con los dispositivos del cliente. Para obtener más información, consulte los siguientes temas:

- Administración de las asociaciones de dispositivos de cliente (AWS CLI)
- Administración de puntos de conexión del dispositivo principal
- Componentes de dispositivo de cliente proporcionados por AWS
- Crear implementaciones

Asociación de los dispositivos de cliente

Para usar la detección en la nube, asocie los dispositivos de cliente a un dispositivo principal para que puedan detectar el dispositivo principal. A continuación, pueden usar la API de detección de Greengrass para recuperar la información de conectividad y los certificados de sus dispositivos principales asociados.

Del mismo modo, desasocie los dispositivos de cliente de un dispositivo principal para evitar que detecten el dispositivo principal.

Temas

- Administración de las asociaciones de dispositivos de cliente (consola)
- Administración de las asociaciones de dispositivos de cliente (AWS CLI)
- Administración de las asociaciones de dispositivos de cliente (API)

Administración de las asociaciones de dispositivos de cliente (consola)

Puede usar la AWS IoT Greengrass consola para ver, agregar y eliminar asociaciones de dispositivos cliente.

Visualización de las asociaciones de dispositivos de cliente de un dispositivo principal (consola)

1. Vaya a la consola de AWS IoT Greengrass.
2. Elija Dispositivos principales.
3. Elija el dispositivo principal que desee administrar.
4. En la página de detalles del dispositivo principal, elija la pestaña Client devices (Dispositivos cliente).
5. En la sección Dispositivos cliente asociados, puede ver qué dispositivos (AWS IoT cosas) cliente están asociados al dispositivo principal.

Asociación de los dispositivos de cliente al dispositivo principal (consola)

1. Vaya a la consola de AWS IoT Greengrass.
2. Elija Dispositivos principales.
3. Elija el dispositivo principal que desee administrar.
4. En la página de detalles del dispositivo principal, elija la pestaña Client devices (Dispositivos cliente).
5. En la sección Associated client devices (Dispositivos cliente asociados), seleccione Associate client devices (Asociar dispositivos cliente).
6. En el modal Associate client devices with core device (Asociar dispositivos cliente a dispositivos principales), haga lo siguiente para cada dispositivo cliente que desee asociar:
   a. Introduzca el nombre del elemento AWS IoT que desee asociar como dispositivo cliente.
   b. Elija Add (Añadir).
7. Elija Associate (Asociar).

   Los dispositivos cliente que asoció ahora pueden usar la API de detección de Greengrass para detectar este dispositivo principal.

Desasociación de dispositivos de cliente de un dispositivo principal (consola)

1. Vaya a la consola de AWS IoT Greengrass.
2. Elija Dispositivos principales.
3. Elija el dispositivo principal que desee administrar.
4. En la página de detalles del dispositivo principal, elija la pestaña Client devices (Dispositivos cliente).
5. En la sección Dispositivos de cliente asociados, elija cada dispositivo de cliente para desasociar.
6. Elija Desasociar.
7. En el cuadro de confirmación, elija Desasociar.

   Los dispositivos de cliente que desasoció ahora no pueden usar la API de detección de Greengrass para detectar este dispositivo principal.

Administración de las asociaciones de dispositivos de cliente (AWS CLI)

Puede usar AWS Command Line Interface (AWS CLI) para administrar las asociaciones de dispositivos cliente de un dispositivo principal.

Visualización de las asociaciones de dispositivos de cliente de un dispositivo principal (AWS CLI)

- Use el siguiente comando: list-client-devices-associated- with-core-device.

Asociación de los dispositivos de cliente al dispositivo principal (AWS CLI)

- Use el siguiente comando: batch-associate-client-device- with-core-device.

Desasociación de los dispositivos de cliente de un dispositivo principal (AWS CLI)

- Use el siguiente comando: batch-disassociate-client-device- from-core-device.

Administración de las asociaciones de dispositivos de cliente (API)

Puede usar la AWS API para administrar las asociaciones de dispositivos cliente de un dispositivo principal.

Para ver las asociaciones de dispositivos cliente de un dispositivo principal (AWS API)

- Utilice la siguiente operación: ListClientDevicesAssociatedWithCoreDevice.

Para asociar dispositivos cliente a un dispositivo principal (AWS API)

- Utilice la siguiente operación: BatchAssociateClientDeviceWithCoreDevice.

Para desasociar los dispositivos cliente de un dispositivo principal (AWS API)

- Utilice la siguiente operación: BatchDisassociateClientDeviceFromCoreDevice.

Autenticación de clientes sin conexión

Con la autenticación sin conexión, puede configurar su dispositivo AWS IoT Greengrass principal para que los dispositivos cliente puedan conectarse a un dispositivo principal, incluso cuando el dispositivo principal no esté conectado a la nube. Cuando utiliza la autenticación sin conexión, sus dispositivos de Greengrass pueden seguir funcionando en un entorno parcialmente desconectado.

Para utilizar la autenticación sin conexión en un dispositivo de cliente con conexión a la nube, necesita lo siguiente:

- Un dispositivo AWS IoT Greengrass Core con el Autenticación del dispositivo de cliente componente implementado. Debe utilizar la versión 2.3.0 o posterior para la autenticación sin conexión.
- Una conexión a la nube para el dispositivo principal durante la conexión inicial de los dispositivos de cliente.

Almacenamiento de credenciales del cliente

Cuando un dispositivo cliente se conecta a un dispositivo principal por primera vez, el dispositivo principal llama al AWS IoT Greengrass servicio. Cuando se llama, Greengrass valida el registro del dispositivo de cliente como un objeto AWS IoT . También valida que el dispositivo tenga un certificado válido. El dispositivo principal almacena esta información de forma local.

La próxima vez que el dispositivo se conecte, el dispositivo principal de Greengrass intentará validar el dispositivo cliente con el AWS IoT Greengrass servicio. Si no se puede conectar AWS IoT

Greengrass, el dispositivo principal utilizará la información del dispositivo almacenada localmente para validar el dispositivo cliente.

Puede configurar el tiempo durante el que el dispositivo principal de Greengrass almacena las credenciales. Puede establecer el tiempo de espera de un minuto a 2 147 483 647 minutos con la opción de configuración `clientDeviceTrustDurationMinutes` de la configuración del componente de autenticación del dispositivo de cliente. El valor predeterminado es un minuto, lo que desactiva de forma efectiva la autenticación sin conexión. Cuando establezca este tiempo de espera, le recomendamos que tenga en cuenta sus necesidades de seguridad. También debe tener en cuenta cuánto tiempo espera que funcionen los dispositivos principales mientras estén desconectados de la nube.

El dispositivo principal actualiza su almacenamiento de credenciales tres veces:

1. Cuando un dispositivo se conecta al dispositivo principal por primera vez.
2. Si el dispositivo principal está conectado a la nube, cuando un dispositivo de cliente se vuelve a conectar al dispositivo principal.
3. Si el dispositivo principal está conectado a la nube, una vez al día para actualizar todo el almacén de credenciales.

Cuando el dispositivo principal de Greengrass actualiza su almacén de credenciales, utiliza la operación. ListClientDevicesAssociatedWithCoreDevice Greengrass solo actualiza los dispositivos devueltos por esta operación. Para asociar los dispositivos de cliente al dispositivo principal, consulte Asociación de los dispositivos de cliente.

Para usar la `ListClientDevicesAssociatedWithCoreDevice` operación, debe añadir el permiso para la operación al rol AWS Identity and Access Management (IAM) asociado a la operación que se está ejecutando. Cuenta de AWS AWS IoT Greengrass Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Administración de puntos de conexión del dispositivo principal

Cuando utiliza la detección en la nube, almacena los puntos de conexión del agente de MQTT para los dispositivos principales en el servicio en la nube de AWS IoT Greengrass . Los dispositivos cliente se conectan AWS IoT Greengrass para recuperar estos puntos finales y otra información para sus dispositivos principales asociados.

Para cada dispositivo principal, puede administrar los puntos de conexión de forma automática o manual.

- Administración automática de los puntos de conexión con un detector de IP

  Puede implementar el componente detector de IP para administrar automáticamente los puntos de conexión de los dispositivos principales si tiene una configuración de red no compleja, por ejemplo, si los dispositivos cliente están en la misma red que el dispositivo principal. No puede utilizar el componente detector de IP si el dispositivo principal está detrás de un enrutador que reenvía el puerto agente de MQTT al dispositivo principal, por ejemplo.

  El componente detector de IP también es útil si se implementa en grupos de objetos, ya que administra los puntos de conexión de todos los dispositivos principales del grupo de objetos. Para obtener más información, consulte Uso del detector de IP para administrar automáticamente los puntos de conexión.

- Administración manual de los puntos de conexión

  Si no puede utilizar el componente detector de IP, debe administrar manualmente los puntos de conexión de los dispositivos principales. Puede actualizar estos puntos de conexión con la consola o la API. Para obtener más información, consulte Administración manual de los puntos de conexión.

Temas

- Uso del detector de IP para administrar automáticamente los puntos de conexión
- Administración manual de los puntos de conexión

Uso del detector de IP para administrar automáticamente los puntos de conexión

Si tiene una configuración de red sencilla, como los dispositivos de cliente en la misma red que el dispositivo principal, puede implementar el componente detector de IP para hacer lo siguiente:

- Supervise la información de conectividad de red local del dispositivo principal de Greengrass. Esta información incluye los puntos de conexión de la red del dispositivo principal y el puerto en el que opera el agente de MQTT.
- Informe la información de conectividad del dispositivo principal al servicio AWS IoT Greengrass en la nube.

El componente detector de IP sobrescribe los puntos de conexión que se configuran manualmente.

> **Important**
>
> La AWS IoT política del dispositivo principal debe permitir el `greengrass:UpdateConnectivityInfo` permiso para usar el componente detector de IP. Para obtener más información, consulte AWS IoT políticas para las operaciones del plano de datos y Configure la política de AWS IoT cosas.

Puede realizar uno de los siguientes procedimientos para implementar el componente detector de IP:

- Utilice la página Configurar la detección en la consola. Para obtener más información, consulte Configuración de la detección en la nube (consola).
- Cree y revise las implementaciones para incluir el detector de IP. Puede usar la consola o la AWS API para administrar las implementaciones. AWS CLI Para obtener más información, consulte Crear implementaciones.

Implementación del componente detector de IP (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, elija la pestaña Componentes públicos y, luego, elija aws.greengrass.clientdevices.IPDetector.
3. En la página aws.greengrass.clientdevices.IPDetector, elija Implementar.
4. En Agregar a la implementación, elija una implementación existente para revisarla o cree una nueva y, a continuación, elija Siguiente.
5. Si opta por crear una nueva implementación, elija el dispositivo principal o el grupo de objetos de destino para la implementación. En la página Especificar el destino, en Destino de la implementación, elija un dispositivo principal o un grupo de objetos y, a continuación, elija Siguiente.
6. En la página Seleccionar componentes, compruebe que el componente aws.greengrass.clientdevices.IPDetector esté seleccionado y elija Siguiente.
7. En la página Configurar componentes, seleccione aws.greengrass.clientdevices.IPDetector y haga lo siguiente:

   a. Seleccione Configurar componente.

   b. En el cuadro Configurar aws.greengrass.clientdevices.IPDetector, en Actualizar configuración, en Configuración para combinar, puede introducir una actualización de

configuración para configurar el componente detector de IP. Puede especificar una de las siguientes opciones de configuración:

- `defaultPort`: (opcional) el puerto del agente MQTT para informar cuando este componente detecta direcciones IP. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883.
- `includeIPv4LoopbackAddrs`— (Opcional) Puede habilitar esta opción para detectar y reportar las direcciones de IPv4 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.
- `includeIPv4LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.
- `includeIPv6LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv6 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv6LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv6 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv4Addrs`: (opcional) el valor predeterminado está establecido en verdadero. Puede activar esta opción para publicar IPv4 las direcciones que se encuentran en el dispositivo principal. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv6Addrs`— (Opcional) Puede activar esta opción para publicar IPv6 las direcciones que se encuentran en el dispositivo principal. Configure `includeIPv4Addrs`

en `false` para usar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.

La actualización de la configuración podría parecerse al siguiente ejemplo.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false
}
```

c. Elija Confirmar para cerrar el cuadro y, a continuación, elija Siguiente.

8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Revisar, elija Implementar.

   La implementación puede tardar hasta un minuto para completarse.

Implementación del componente detector de IP (AWS CLI)

Para implementar el componente detector de IP, cree un documento de implementación que incluya `aws.greengrass.clientdevices.IPDetector` en el objeto `components` y especifique la actualización de configuración del componente. Siga las instrucciones en Crear implementaciones para crear una implementación nueva o revisar una implementación existente.

Puede especificar cualquiera de las siguientes opciones para configurar el componente detector de IP cuando cree el documento de implementación:

- `defaultPort`: (opcional) el puerto del agente MQTT para informar cuando este componente detecta direcciones IP. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883.
- `includeIPv4LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv4 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.
- `includeIPv4LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo

principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.

- `includeIPv6LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv6 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv6LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv6 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv4Addrs`: (opcional) el valor predeterminado está establecido en verdadero. Puede activar esta opción para publicar IPv4 las direcciones que se encuentran en el dispositivo principal. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv6Addrs`— (Opcional) Puede activar esta opción para publicar IPv6 las direcciones que se encuentran en el dispositivo principal. Configure `includeIPv4Addrs` en `false` para usar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.

El siguiente ejemplo de documento de implementación parcial especifica declarar el puerto 8883 como puerto agente de MQTT.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Administración manual de los puntos de conexión

Puede administrar manualmente los puntos de conexión del agente de MQTT para los dispositivos principales.

Cada punto de conexión del agente de MQTT tiene la siguiente información:

Punto de conexión (`HostAddress`)

Una dirección IP o una dirección DNS donde los dispositivos de cliente pueden conectarse a un agente de MQTT en el dispositivo principal.

Port (`PortNumber`)

El puerto en el que opera el agente de MQTT en el dispositivo principal.

Puede configurar este puerto en el componente agente de MQTT de Moquette, que utiliza de forma predeterminada el puerto 8883.

Metadatos (`Metadata`)

Metadatos adicionales para proporcionarlos a los dispositivos de cliente que se conectan a este punto de conexión.

Temas

- Administración de los puntos de conexión (consola)
- Administración de puntos de conexión (AWS CLI)
- Administración de puntos de conexión (API)

Administración de los puntos de conexión (consola)

Puede usar la AWS IoT Greengrass consola para ver, actualizar y eliminar los puntos finales de un dispositivo principal.

Cómo administrar los puntos de conexión de un dispositivo principal (consola)

1. Vaya a la consola de AWS IoT Greengrass.
2. Elija Dispositivos principales.
3. Elija el dispositivo principal que desee administrar.

4. En la página de detalles del dispositivo principal, elija la pestaña Client devices (Dispositivos cliente).
5. En la sección de puntos de conexión del agente de MQTT, puede ver los puntos de conexión del agente de MQTT del dispositivo principal. Seleccione Administrar puntos de conexión.
6. En el cuadro Administrar puntos de conexión, agregue o elimine los puntos de conexión del agente de MQTT para el dispositivo principal.
7. Elija Actualizar.

Administración de puntos de conexión (AWS CLI)

Puede usar AWS Command Line Interface (AWS CLI) para administrar los puntos finales de un dispositivo principal.

> **Note**
>
> Como la compatibilidad con dispositivos cliente AWS IoT Greengrass V2 es retrocompatible AWS IoT Greengrass V1, puedes utilizar AWS IoT Greengrass V2 nuestras operaciones de AWS IoT Greengrass V1 API para gestionar los puntos finales principales de los dispositivos.

Cómo obtener puntos de conexión para un dispositivo principal (AWS CLI)

- Ejecute cualquiera de los siguientes comandos:
  - greengrassv2: get-connectivity-info
  - hierba verde: get-connectivity-info

Cómo actualizar los puntos de conexión de un dispositivo principal (AWS CLI)

- Ejecute cualquiera de los siguientes comandos:
  - hierba verde v2: update-connectivity-info
  - hierba verde: update-connectivity-info

Administración de puntos de conexión (API)

Puede usar la AWS API para administrar los puntos finales de un dispositivo principal.

> **Note**
>
> Como la compatibilidad con dispositivos cliente AWS IoT Greengrass V2 es retrocompatible AWS IoT Greengrass V1, puedes utilizar AWS IoT Greengrass V2 nuestras operaciones de AWS IoT Greengrass V1 API para gestionar los puntos finales principales de los dispositivos.

Para obtener puntos finales para un dispositivo principal (API)AWS

- Ejecute cualquiera de las siguientes operaciones:
  - V2: GetConnectivityInfo
  - V1: GetConnectivityInfo

Para actualizar los puntos finales de un dispositivo principal (AWS API)

- Ejecute cualquiera de las siguientes operaciones:
  - V2: UpdateConnectivityInfo
  - V1: UpdateConnectivityInfo

Elección de un agente MQTT

AWS IoT Greengrass ofrece opciones para que pueda elegir qué agente MQTT local desea ejecutar en sus dispositivos principales. Los dispositivos de cliente se conectan al agente MQTT que se ejecuta en un dispositivo principal, así que elija un agente MQTT que sea compatible con los dispositivos de cliente a los que desee conectarse.

> **Note**
>
> Se recomienda implementar solo un componente agente MQTT. Los componentes puente de MQTT y detector de IP solo funcionan con un componente agente MQTT a la vez. Si implementa varios componentes agente de MQTT, debe configurarlos para que usen puertos diferentes.

Puede elegir entre los siguientes agentes MQTT:

- Agente MQTT 3.1.1 (Moquette): `aws.greengrass.clientdevices.mqtt.Moquette`

Elija esta opción para un agente MQTT ligero que cumpla con el estándar MQTT 3.1.1. El agente MQTT AWS IoT Core y SDK para dispositivos con AWS IoT también son compatibles con el estándar MQTT 3.1.1, por lo que puede usar estas características para crear una aplicación que use MQTT 3.1.1 en sus dispositivos y en la Nube de AWS.

- Agente MQTT 5 (EMQX): `aws.greengrass.clientdevices.mqtt.EMQX`

  Elija esta opción para usar las características de MQTT 5 en la comunicación entre los dispositivos principales y los dispositivos de cliente. Este componente usa más recursos que el agente MQTT 3.1.1 de Moquette y, en los dispositivos principales de Linux, requiere Docker.

  MQTT 5 es compatible con versiones anteriores de MQTT 3.1.1, por lo que puede conectar dispositivos de cliente que usen MQTT 3.1.1 a este agente. Si usa el agente MQTT 3.1.1 de Moquette, puede sustituirlo por el agente MQTT 5 de EMQX y los dispositivos de cliente podrán seguir conectándose y funcionando como de costumbre.

- Implementación de un agente personalizado

  Elija esta opción para crear un componente de agente local personalizado para comunicarse con los dispositivos de cliente. Puede crear un agente local personalizado que use un protocolo distinto de MQTT. AWS IoT Greengrass proporciona un SDK de componentes que puede usar para autenticar y autorizar los dispositivos de cliente. Para obtener más información, consulte Úselo SDK para dispositivos con AWS IoT para comunicarse con el núcleo de Greengrass, otros componentes y AWS IoT Core y Autenticación y autorización de los dispositivos de cliente.

Conexión de dispositivos cliente a un dispositivo AWS IoT Greengrass Core con un intermediario MQTT

Cuando utiliza un intermediario MQTT en su dispositivo AWS IoT Greengrass Core, el dispositivo utiliza una autoridad de certificación (CA) del dispositivo principal exclusiva del dispositivo para emitir un certificado al agente a fin de establecer conexiones TLS mutuas con los clientes.

AWS IoT Greengrass generará automáticamente una CA para el dispositivo principal, o puede proporcionar la suya propia. La CA del dispositivo principal está registrada AWS IoT Greengrass cuando se conecta el Autenticación del dispositivo de cliente componente. La CA del dispositivo principal generada automáticamente es persistente, el dispositivo seguirá usando la misma CA mientras el componente de autenticación del dispositivo de cliente esté configurado.

Cuando el agente MQTT se inicia, solicita un certificado. El componente de autenticación del dispositivo de cliente emite un certificado X.509 mediante la CA del dispositivo principal. El certificado se rota cuando el agente se inicia, cuando el certificado caduca o cuando cambia la información de conectividad, como la dirección IP. Para obtener más información, consulte Rotación de certificados en el agente MQTT local.

Para conectar un cliente al agente MQTT, necesita lo siguiente:

- El dispositivo cliente debe tener la CA del dispositivo AWS IoT Greengrass principal. Puede obtener esta CA mediante la detección en la nube o proporcionándola manualmente. Para obtener más información, consulte Uso de su propia autoridad de certificación.
- El nombre de dominio completamente calificado (FQDN) o la dirección IP del dispositivo principal debe estar presente en el certificado del agente emitido por la CA del dispositivo principal. Para garantizar esto, use el componente Detector de IP o configure manualmente la dirección IP. Para obtener más información, consulte Administración de puntos de conexión del dispositivo principal.
- El componente de autenticación del dispositivo de cliente debe dar permiso al dispositivo de cliente para conectarse al dispositivo principal de Greengrass. Para obtener más información, consulte Autenticación del dispositivo de cliente.

Uso de su propia autoridad de certificación

Si los dispositivos de cliente no pueden acceder a la nube para detectar su dispositivo principal, puede proporcionar una autoridad de certificación (CA) del dispositivo principal. Su dispositivo principal de Greengrass usa la CA del dispositivo principal para emitir certificados para su agente de MQTT. Una vez que configure el dispositivo principal y suministre su CA al dispositivo de cliente, sus dispositivos de cliente pueden conectarse al punto de conexión y verificar el protocolo de enlace TLS mediante la CA del dispositivo principal (la CA proporcionada por usted o generada automáticamente).

Para configurar el componente Autenticación del dispositivo de cliente para que use la CA del dispositivo principal, defina el parámetro de configuración `certificateAuthority` al implementar el componente. Debe proporcionar los siguientes detalles durante la configuración:

- La ubicación del certificado de CA de un dispositivo principal.
- La clave privada del certificado de CA del dispositivo principal.
- (Opcional) La cadena de certificados al certificado raíz si la CA del dispositivo principal es una CA intermedia.

Si proporciona una CA del dispositivo principal, AWS IoT Greengrass registra la CA en la nube.

Puede almacenar sus certificados en un módulo de seguridad de hardware o en el sistema de archivos. En el siguiente ejemplo, se muestra una configuración `certificateAuthority` para una CA intermedia almacenada mediante HSM/TPM. Tenga en cuenta que la cadena de certificados solo se puede almacenar en el disco.

```
"certificateAuthority": {
    "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
    "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

En este ejemplo, el parámetro de configuración `certificateAuthority` configura el componente de autenticación del dispositivo de cliente para que use una CA intermedia del sistema de archivos:

```
"certificateAuthority": {
    "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",
    "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

Para conectar los dispositivos a su dispositivo AWS IoT Greengrass principal, haga lo siguiente:

1. Cree una entidad de certificación (CA) intermedia para el dispositivo principal de Greengrass usando la CA raíz de su organización. Se recomienda usar una CA intermedia como práctica recomendada de seguridad.
2. Proporcione el certificado CA intermedio, la clave privada y la cadena de certificados a su CA raíz al dispositivo principal de Greengrass. Para obtener más información, consulte Autenticación del dispositivo de cliente. La CA intermedia se convierte en la CA del dispositivo principal del dispositivo principal de Greengrass, y el dispositivo registra la CA en ella. AWS IoT Greengrass
3. Registre el dispositivo cliente como una AWS IoT cosa. Para obtener más información, consulte Crear un objeto en la Guía para desarrolladores de AWS IoT Core . Agregue la clave privada, la clave pública, el certificado del dispositivo y el certificado de CA raíz al dispositivo de cliente. La forma de agregar la información depende del dispositivo y el software.

Una vez que configure su dispositivo, podrá usar el certificado y la cadena de claves pública para conectarse al dispositivo principal de Greengrass. Su software es responsable de encontrar los

puntos de conexión del dispositivo principal. Puede configurar el punto de conexión manualmente para el dispositivo principal. Para obtener más información, consulte Administración manual de los puntos de conexión.

Prueba de las comunicaciones del dispositivo de cliente

Los dispositivos cliente pueden SDK para dispositivos con AWS IoT utilizarla para detectar, conectarse y comunicarse con un dispositivo principal. Puede usar el cliente de descubrimiento de Greengrass SDK para dispositivos con AWS IoT para usar la API de descubrimiento de Greengrass, que devuelve información sobre los dispositivos principales a los que se puede conectar un dispositivo cliente. La respuesta de la API incluye los puntos de conexión del agente MQTT para conectarse y los certificados que se utilizan para verificar la identidad de cada dispositivo principal. A continuación, el dispositivo de cliente puede probar cada punto de conexión hasta que se conecte correctamente a un dispositivo principal.

Los dispositivos de cliente solo pueden detectar los dispositivos principales a los que se asocien. Antes de probar las comunicaciones entre un dispositivo de cliente y un dispositivo principal, debe asociar el dispositivo de cliente al dispositivo principal. Para obtener más información, consulte Asociación de los dispositivos de cliente.

La API de detección de Greengrass devuelve los puntos de conexión del agente MQTT del dispositivo principal que especifique. Puede usar el componente detector de IP para administrar estos puntos de conexión por usted, o puede administrarlos manualmente para cada dispositivo principal. Para obtener más información, consulte Administración de puntos de conexión del dispositivo principal.

> **Note**
>
> Para usar la API de detección de Greengrass, un dispositivo de cliente debe tener el permiso `greengrass:Discover`. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos cliente.

SDK para dispositivos con AWS IoT Está disponible en varios lenguajes de programación. Para obtener más información, consulte AWS IoT Dispositivo SDKs en la Guía para AWS IoT Core desarrolladores.

Temas

- Prueba de comunicaciones (Python)

- Prueba de las comunicaciones (C++)
- Pruebe las comunicaciones (JavaScript)
- Prueba de comunicaciones (Java)

Prueba de comunicaciones (Python)

En esta sección, utilizará un ejemplo de detección de Greengrass en SDK para dispositivos con AWS IoT versión 2 para Python para probar las comunicaciones entre un dispositivo de cliente y un dispositivo principal.

> **Important**
>
> Para usar la SDK para dispositivos con AWS IoT versión 2 para Python, el dispositivo debe ejecutar Python 3.6 o una versión posterior.

Para probar las comunicaciones (SDK para dispositivos con AWS IoT v2 para Python)

1. Descargue e instale la SDK para dispositivos con AWS IoT versión 2 para Python en el AWS IoT dispositivo para conectarlo como dispositivo cliente.

   En el dispositivo de cliente, haga lo siguiente:

   a. Clona el repositorio SDK para dispositivos con AWS IoT v2 para Python para descargarlo.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Instale la SDK para dispositivos con AWS IoT versión 2 para Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Cambie a la carpeta de muestras de la SDK para dispositivos con AWS IoT versión 2 para Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples/greengrass
   ```

3. Ejecute la aplicación de ejemplo de detección de Greengrass. Esta aplicación espera argumentos que especifican el nombre del objeto del dispositivo de cliente, el tema y el mensaje de MQTT que se van a usar y los certificados que autentican y protegen la conexión. En el

siguiente ejemplo, se envía el mensaje “Hello World” al tema `clients/`*`MyClientDevice1`*`/hello/world`.

- *`MyClientDevice1`*Sustitúyala por el nombre del dispositivo cliente.
- *`~/certs/AmazonRootCA1.pem`*Sustitúyalo por la ruta al certificado de CA raíz de Amazon en el dispositivo cliente.
- *`~/certs/device.pem.crt`*Sustitúyalo por la ruta al certificado del dispositivo del dispositivo cliente.
- *`~/certs/private.pem.key`*Sustitúyalo por la ruta al archivo de clave privada del dispositivo cliente.
- *`us-east-1`*Sustitúyala por la AWS región en la que funcionan el dispositivo cliente y el dispositivo principal.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

La aplicación de ejemplo de detección envía el mensaje 10 veces y se desconecta. También se suscribe al mismo tema en el que publica los mensajes. Si el resultado indica que la aplicación recibió mensajes MQTT sobre el tema, el dispositivo de cliente puede comunicarse correctamente con el dispositivo principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
```

```
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

También puede ver los registros de Greengrass en el dispositivo principal para comprobar si el dispositivo de cliente se conecta y envía mensajes correctamente. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Prueba de las comunicaciones (C++)

En esta sección, utilizará un ejemplo de detección de Greengrass en el SDK para dispositivos con AWS IoT versión 2 para C++ a fin de probar las comunicaciones entre un dispositivo de cliente y un dispositivo principal.

Para compilar la SDK para dispositivos con AWS IoT versión 2 para C++, un dispositivo debe tener las siguientes herramientas:

- C++ 11 o posterior
- CMake 3.1 o una versión posterior
- Uno de los siguientes copiladores:

- GCC 4.8 o posterior
- Clang 3.9 o posterior
- MSVC 2015 o posterior

Para probar las comunicaciones (SDK para dispositivos con AWS IoT v2 para C++)

1. Descargue y cree la SDK para dispositivos con AWS IoT versión 2 para C++ en AWS IoT el dispositivo que desee conectar como dispositivo cliente.

   En el dispositivo de cliente, haga lo siguiente:

   a. Cree una carpeta para el espacio de trabajo SDK para dispositivos con AWS IoT de la versión 2 para C++ y cámbiela a ella.

   ```
   cd
   mkdir iot-device-sdk-cpp
   cd iot-device-sdk-cpp
   ```

   b. Clona el SDK para dispositivos con AWS IoT repositorio de la versión 2 para C++ para descargarlo. El indicador `--recursive` especifica la descarga de submódulos.

   ```
   git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
   ```

   c. Cree una carpeta para el resultado de la SDK para dispositivos con AWS IoT compilación de la versión 2 para C++ y cámbiela a ella.

   ```
   mkdir aws-iot-device-sdk-cpp-v2-build
   cd aws-iot-device-sdk-cpp-v2-build
   ```

   d. Compila la SDK para dispositivos con AWS IoT v2 para C++.

   ```
   cmake -DCMAKE_INSTALL_PREFIX="~/iot-device-sdk-cpp" -
   DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
   cmake --build . --target install
   ```

2. Cree la aplicación de ejemplo Greengrass Discovery en la SDK para dispositivos con AWS IoT versión 2 para C++. Haga lo siguiente:

   a. Cambie a la carpeta de ejemplos de Greengrass Discovery en la SDK para dispositivos con AWS IoT versión 2 para C++.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

b. Cree una carpeta para el resultado de la compilación de ejemplo de detección de Greengrass y cambie a ella.

```
mkdir build
cd build
```

c. Cree la aplicación de ejemplo de detección de Greengrass.

```
cmake -DCMAKE_PREFIX_PATH="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ..
cmake --build . --config "Release"
```

3. Ejecute la aplicación de ejemplo de detección de Greengrass. Esta aplicación espera argumentos que especifiquen el nombre del dispositivo de cliente, el tema de MQTT que se va a utilizar y los certificados que autentican y protegen la conexión. En el siguiente ejemplo, se suscribe al tema `clients/`*`MyClientDevice1`*`/hello/world` y se publica un mensaje sobre el mismo tema que se introduce en la línea de comandos.

   - *`MyClientDevice1`*Sustitúyala por el nombre del dispositivo cliente.
   - *`~/certs/AmazonRootCA1.pem`*Sustitúyalo por la ruta al certificado de CA raíz de Amazon en el dispositivo cliente.
   - *`~/certs/device.pem.crt`*Sustitúyalo por la ruta al certificado del dispositivo del dispositivo cliente.
   - *`~/certs/private.pem.key`*Sustitúyalo por la ruta al archivo de clave privada del dispositivo cliente.
   - *`us-east-1`*Sustitúyala por la AWS región en la que funcionan el dispositivo cliente y el dispositivo principal.

```
./basic-discovery \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1
```

La aplicación de ejemplo de detección se suscribe al tema y le pide que introduzca un mensaje para publicarlo.

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn
 arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to
 203.0.113.0:8883
Successfully subscribed to clients/MyClientDevice1/hello/world
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

4. Introduzca un mensaje, como **`Hello World!`**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
Hello World!
```

Si el resultado indica que la aplicación recibió el mensaje MQTT sobre el tema, el dispositivo de cliente puede comunicarse correctamente con el dispositivo principal.

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

También puede ver los registros de Greengrass en el dispositivo principal para comprobar si el dispositivo de cliente se conecta y envía mensajes correctamente. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Pruebe las comunicaciones (JavaScript)

En esta sección, utilizará un ejemplo de descubrimiento de Greengrass en la SDK para dispositivos con AWS IoT versión 2 JavaScript para probar las comunicaciones entre un dispositivo cliente y un dispositivo principal.

> **Important**
>
> Para usar la SDK para dispositivos con AWS IoT versión 2 JavaScript, un dispositivo debe ejecutar Node v10.0 o una versión posterior.

Para probar las comunicaciones (SDK para dispositivos con AWS IoT v2 para) JavaScript

1. Descargue e instale la SDK para dispositivos con AWS IoT versión 2 JavaScript para el AWS IoT dispositivo que desee conectar como dispositivo cliente.

   En el dispositivo de cliente, haga lo siguiente:

   a. Clona la SDK para dispositivos con AWS IoT v2 para el JavaScript repositorio para descargarla.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

   b. Instale la SDK para dispositivos con AWS IoT v2 para JavaScript.

   ```
   cd aws-iot-device-sdk-js-v2
   npm install
   ```

2. Cambie a la carpeta de ejemplos de Greengrass discovery en la SDK para dispositivos con AWS IoT versión 2 para. JavaScript

   ```
   cd samples/node/greengrass/basic_discovery
   ```

3. Instale la aplicación de ejemplo de detección de Greengrass.

   ```
   npm install
   ```

4. Ejecute la aplicación de ejemplo de detección de Greengrass. Esta aplicación espera argumentos que especifican el nombre del objeto del dispositivo de cliente, el tema y el mensaje de MQTT que se van a usar y los certificados que autentican y protegen la conexión. En el siguiente ejemplo, se envía el mensaje “Hello World” al tema `clients/`*`MyClientDevice1`*`/hello/world`.

   - *`MyClientDevice1`*Sustitúyalo por el nombre del dispositivo cliente.

- *~/certs/AmazonRootCA1.pem*Sustitúyalo por la ruta al certificado de CA raíz de Amazon en el dispositivo cliente.
- *~/certs/device.pem.crt*Sustitúyalo por la ruta al certificado del dispositivo del dispositivo cliente.
- *~/certs/private.pem.key*Sustitúyalo por la ruta al archivo de clave privada del dispositivo cliente.
- *us-east-1*Sustitúyala por la AWS región en la que funcionan el dispositivo cliente y el dispositivo principal.

```
node dist/index.js \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --message 'Hello World!' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

La aplicación de ejemplo de detección envía el mensaje 10 veces y se desconecta. También se suscribe al mismo tema en el que publica los mensajes. Si el resultado indica que la aplicación recibió mensajes MQTT sobre el tema, el dispositivo de cliente puede comunicarse correctamente con el dispositivo principal.

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-
MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore","connectivity":
[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificat
["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
Trying
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
 fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
 signals thrown, you may want to install a signal trap in your application layer.
Connected to
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
```

```
{"message":"Hello World!","sequence":1}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":2}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":4}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

También puede ver los registros de Greengrass en el dispositivo principal para comprobar si el dispositivo de cliente se conecta y envía mensajes correctamente. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

Prueba de comunicaciones (Java)

En esta sección, utilizará un ejemplo de detección de Greengrass en el SDK para dispositivos con AWS IoT versión 2 para Java para probar las comunicaciones entre un dispositivo de cliente y un dispositivo principal.

> **Important**
>
> Para compilar la SDK para dispositivos con AWS IoT versión 2 para Java, un dispositivo debe tener las siguientes herramientas:
>
> - Java 8 o posterior, con `JAVA_HOME` apuntando a la carpeta de Java.
> - Apache Maven

Para probar las comunicaciones (SDK para dispositivos con AWS IoT v2 para Java)

1. Descargue y cree la SDK para dispositivos con AWS IoT versión 2 para Java en el AWS IoT dispositivo que desee conectar como dispositivo cliente.

   En el dispositivo de cliente, haga lo siguiente:

   a. Clona el SDK para dispositivos con AWS IoT repositorio de la versión 2 para Java para descargarlo.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
   ```

   b. Cambie a la carpeta SDK para dispositivos con AWS IoT v2 para Java.

   c. Compila la SDK para dispositivos con AWS IoT v2 para Java.

   ```
   cd aws-iot-device-sdk-java-v2
   mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
   mvn clean install
   ```

2. Ejecute la aplicación de ejemplo de detección de Greengrass. Esta aplicación espera argumentos que especifiquen el nombre del dispositivo de cliente, el tema de MQTT que se va a utilizar y los certificados que autentican y protegen la conexión. En el siguiente ejemplo, se suscribe al tema `clients/`*`MyClientDevice1`*`/hello/world` y se publica un mensaje sobre el mismo tema que se introduce en la línea de comandos.

   - Sustituya ambas instancias *`MyClientDevice1`* de por el nombre del dispositivo cliente.
   - *`$HOME/certs/AmazonRootCA1.pem`*Sustitúyalo por la ruta al certificado de CA raíz de Amazon en el dispositivo cliente.
   - *`$HOME/certs/device.pem.crt`*Sustitúyalo por la ruta al certificado del dispositivo del dispositivo cliente.

- *$HOME/certs/private.pem.key*Sustitúyalo por la ruta al archivo de clave privada del dispositivo cliente.
- *us-east-1*Sustitúyalo por el Región de AWS lugar donde funcionan el dispositivo cliente y el dispositivo principal.

```
DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file $HOME/certs/AmazonRootCA1.pem \
  --cert $HOME/certs/device.pem.crt \
  --key $HOME/certs/private.pem.key \
  --region us-east-1"

mvn exec:java -pl samples/Greengrass/Discovery \
  -Dexec.mainClass=greengrass.BasicDiscovery \
  -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

La aplicación de ejemplo de detección se suscribe al tema y le pide que introduzca un mensaje para publicarlo.

```
Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing
 arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Started a clean session
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

3. Introduzca un mensaje, como **`Hello World!`**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
Hello World!
```

Si el resultado indica que la aplicación recibió el mensaje MQTT sobre el tema, el dispositivo de cliente puede comunicarse correctamente con el dispositivo principal.

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

También puede ver los registros de Greengrass en el dispositivo principal para comprobar si el dispositivo de cliente se conecta y envía mensajes correctamente. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

API de descubrimiento de Greengrass RESTful

AWS IoT Greengrass proporciona la operación de `Discover` API que los dispositivos cliente pueden usar para identificar los dispositivos principales de Greengrass a los que se pueden conectar. Los dispositivos cliente utilizan esta operación de plano de datos para recuperar la información necesaria para conectarse a los dispositivos principales de Greengrass, donde se asocian a la operación de la BatchAssociateClientDeviceWithCoreDeviceAPI. Cuando un dispositivo cliente se conecta a Internet, puede conectarse al servicio AWS IoT Greengrass en la nube y usar la API de detección para encontrar:

- La dirección IP y el puerto de cada dispositivo principal de Greengrass asociado.
- El certificado de CA del dispositivo principal, que los dispositivos de cliente pueden utilizar para autenticar el dispositivo principal de Greengrass.

> **Note**
>
> Los dispositivos cliente también pueden usar el cliente de detección SDK para dispositivos con AWS IoT para descubrir la información de conectividad de los dispositivos principales de Greengrass. El cliente de detección utiliza la API de detección. Para obtener más información, consulte los siguientes temas:
>
> - Prueba de las comunicaciones del dispositivo de cliente
> - Greengrass Discovery RESTful API en la Guía para AWS IoT Greengrass Version 1 desarrolladores.

Para utilizar esta operación de API, envíe solicitudes HTTP a la API de detección en el punto de conexión del plano de datos de Greengrass. Este punto de conexión de la API tiene el siguiente formato.

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Para obtener una lista de los puntos finales Regiones de AWS y los puntos de conexión compatibles con la API de AWS IoT Greengrass descubrimiento, consulte los AWS IoT Greengrass V2 puntos finales y las cuotas en. Referencia general de AWS Esta operación de la API solo está disponible en el punto de conexión del plano de datos de Greengrass. El punto de conexión del plano de control que se utiliza para administrar los componentes y las implementaciones es diferente del punto de conexión del plano de datos.

> **Note**
>
> La API de descubrimiento es la misma para AWS IoT Greengrass V1 y. AWS IoT Greengrass V2 Si tiene dispositivos cliente que se conectan a un AWS IoT Greengrass V1 núcleo, puede conectarlos a los dispositivos AWS IoT Greengrass V2 principales sin cambiar el código de los dispositivos cliente. Para obtener más información, consulte Greengrass Discovery RESTful API en la Guía para AWS IoT Greengrass Version 1 desarrolladores.

Temas

- Autenticación y autorización de detección
- Solicitud
- Respuesta
- Prueba de la API de detección con cURL

Autenticación y autorización de detección

Para usar la API de detección para recuperar información de conectividad, un dispositivo de cliente debe utilizar la autenticación mutua TLS con un certificado de cliente X.509 para autenticarse. Para obtener más información, consulte Certificados de cliente X.509 en la Guía para desarrolladores de AWS IoT Core .

Un dispositivo de cliente también debe tener permiso para ejecutar la acción `greengrass:Discover`. El siguiente ejemplo AWS IoT de política permite AWS IoT que una cosa llamada `MyClientDevice1` funcione `Discover` por sí misma.

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

Important

Las variables de política de objetos (`iot:Connection.Thing.*`) no son compatibles con las políticas de AWS IoT para dispositivos principales ni operaciones del plano de datos de Greengrass. En su lugar, puede utilizar un comodín que haga coincidir varios dispositivos con nombres similares. Por ejemplo, puede especificar `MyGreengrassDevice*` para que coincida con `MyGreengrassDevice1`, `MyGreengrassDevice2`, etc.

Para obtener más información, consulte Políticas de AWS IoT Core en la Guía para desarrolladores de AWS IoT Core .

Solicitud

La solicitud contiene los encabezados HTTP estándar y se envía al punto de conexión de detección de Greengrass, como se muestra en los ejemplos siguientes.

El número de puerto depende de si el dispositivo principal está configurado para enviar el tráfico HTTPS a través del puerto 8443 o el puerto 443. Para obtener más información, consulte the section called “Realizar la conexión en el puerto 443 o a través de un proxy de red”.

Note

Estos ejemplos utilizan el punto de conexión de Amazon Trust Services (ATS), que se utiliza con los certificados de CA raíz de ATS recomendados. Los puntos de enlace deben coincidir con el tipo de certificado de CA raíz.

Puerto 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/
thing/thing-name
```

Puerto 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/
thing/thing-name
```

> **Note**
>
> Los clientes que se conecten por el puerto 443 deben implementar la extensión TLS de Negociación de Protocolo de Capa de Aplicación (ALPN) y pasar `x-amzn-http-ca` como el `ProtocolName` en el `ProtocolNameList`. Para obtener más información, consulte Protocolos en la Guía para desarrolladores de AWS IoT .

Respuesta

En caso de éxito, el encabezado de la respuesta incluye el código de estado HTTP 200 y el cuerpo de la respuesta contiene el documento de respuesta de la detección.

> **Note**
>
> Como AWS IoT Greengrass V2 utiliza la misma API de descubrimiento que AWS IoT Greengrass V1, la respuesta organiza la información según AWS IoT Greengrass V1 conceptos, como los grupos de Greengrass. La respuesta contiene una lista de grupos de Greengrass. En AWS IoT Greengrass V2, cada dispositivo principal está en su propio grupo, donde el grupo contiene solo ese dispositivo principal y su información de conectividad.

Ejemplo de documentos de respuesta de detección

En el siguiente documento, se muestran la respuesta de un dispositivo de cliente que está asociado a un dispositivo principales de Greengrass. El dispositivo principal tiene un punto de conexión y un certificado de la CA.

```
{
```

```
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

En el siguiente documento, se muestra la respuesta de un dispositivo de cliente que está asociado a dos dispositivos principales. Los dispositivos principales tienen varios puntos de conexión y varios certificados de la CA del grupo.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
```

```
              "hostAddress": "core-device-01-address-2",
              "portNumber": core-device-01-port-2,
              "metadata": "core-device-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
      "Cores": [
        {
          "thingArn":"core-device-02-thing-arn",
          "Connectivity" : [
            {
              "id": "core-device-02-connection-id",
              "hostAddress": "core-device-02-address",
              "portNumber": core-device-02-port,
              "metadata": "core-device-02-connection-1-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

Prueba de la API de detección con cURL

Si instaló cURL, puede probar la API de detección. En el siguiente ejemplo, se especifican los certificados de un dispositivo de cliente para autenticar una solicitud en el punto de conexión de la API de detección de Greengrass.

```
curl -i \
  --cert 1a23bc4d56.cert.pem \
  --key 1a23bc4d56.private.key \
  https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

> **Note**
>
> El argumento `-i` especifica la salida de los encabezados de respuesta HTTP. Puede utilizar esta opción para ayudar a identificar los errores.

Si la solicitud tiene éxito, este comando genera una respuesta similar al siguiente ejemplo.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "Connectivity": [
            {
              "Id": "AUTOIP_192.168.1.4_1",
              "HostAddress": "192.168.1.5",
              "PortNumber": 8883,
              "Metadata": ""
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----\ncert-contents\n-----END CERTIFICATE-----\n"
      ]
    }
  ]
}
```

Si el comando genera un error, consulte Solución de problemas de detección de Greengrass.

Retransmitir mensajes MQTT entre dispositivos cliente y AWS IoT Core

Puede retransmitir mensajes MQTT y otros datos entre dispositivos de cliente y AWS IoT Core. Los dispositivos de cliente se conectan al componente agente de MQTT que se ejecuta en el dispositivo principal. De forma predeterminada, los dispositivos principales no transmiten mensajes o datos MQTT entre los dispositivos cliente y. AWS IoT Core De forma predeterminada, los dispositivos de cliente solo pueden comunicarse entre sí a través de MQTT.

Para retransmitir mensajes MQTT entre dispositivos cliente y AWS IoT Core configurar el componente de puente MQTT para que haga lo siguiente:

- Retransmita mensajes desde los dispositivos cliente a. AWS IoT Core
- AWS IoT Core Retransmita mensajes desde los dispositivos cliente.

> **Note**
>
> El puente MQTT usa QoS 1 para publicar y AWS IoT Core suscribirse, incluso cuando un dispositivo cliente usa QoS 0 para publicar y suscribirse al broker MQTT local. Como resultado, es posible que observe una latencia adicional al retransmitir mensajes MQTT desde los dispositivos cliente del broker MQTT local. AWS IoT Core Para obtener más información acerca de la configuración de MQTT en los dispositivos principales, consulte Configuración de los tiempos de espera y los ajustes de caché de MQTT.

Temas

- Configuración e implementación del componente puente de MQTT
- Retransmisión de mensajes MQTT

Configuración e implementación del componente puente de MQTT

El componente puente de MQTT utiliza una lista de asignaciones de temas, cada una de las cuales especifica un origen y un destino del mensaje. Para retransmitir mensajes entre los dispositivos cliente AWS IoT Core, implementar el componente MQTT bridge y especificar cada tema de origen y destino en la configuración del componente.

Para implementar el componente puente MQTT en un dispositivo principal o en un grupo de dispositivos principales, cree una implementación que incluya el componente `aws.greengrass.clientdevices.mqtt.Bridge`. Especifique las asignaciones de temas, `mqttTopicMapping`, en la configuración del componente puente de MQTT en la implementación.

El siguiente ejemplo define una implementación que configura el componente puente de MQTT para retransmitir mensajes sobre temas `clients/+/hello/world` que coinciden con el filtro de temas desde los dispositivos de cliente a AWS IoT Core. La actualización de configuración `merge` requiere un objeto JSON serializado. Para obtener más información, consulte Actualización de las configuraciones de los componentes.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/
+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
    ...
  }
}
```

Retransmisión de mensajes MQTT

Para retransmitir mensajes MQTT entre dispositivos cliente y configurar e AWS IoT Core implementar el componente MQTT Bridge y especificar los temas que se van a retransmitir.

Example Ejemplo: retransmitir mensajes sobre un tema desde los dispositivos cliente a AWS IoT Core

El siguiente ejemplo define una implementación que configura el componente puente de MQTT para retransmitir mensajes sobre temas `clients/+/hello/world/event` que coinciden con el filtro de temas desde los dispositivos de cliente a AWS IoT Core.

```
{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example Ejemplo: retransmitir mensajes sobre un tema desde AWS IoT Core los dispositivos cliente

El siguiente ejemplo define una implementación que configura el componente puente de MQTT para retransmitir mensajes sobre temas `clients/+/hello/world/event/response` que coinciden con el filtro de temas desde AWS IoT Core a los dispositivos de cliente.

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

Interacción con los dispositivos de cliente en los componentes

Puede desarrollar componentes personalizados de Greengrass que interactúen con los dispositivos de cliente conectados a un dispositivo principal. Por ejemplo, puede desarrollar componentes que hagan lo siguiente:

- Actuar en función de los mensajes MQTT de los dispositivos de cliente y enviar datos a los destinos de Nube de AWS .
- Enviar mensajes MQTT a los dispositivos de cliente para iniciar acciones.

Los dispositivos de cliente se conectan y se comunican con un dispositivo principal a través del componente agente MQTT que se ejecuta en el dispositivo principal. De forma predeterminada, los dispositivos de cliente solo pueden comunicarse entre sí a través de MQTT y los componentes de Greengrass no pueden recibir estos mensajes MQTT ni enviar mensajes a los dispositivos de cliente.

Los componentes de Greengrass utilizan la interfaz local de publicación/suscripción para comunicarse en un dispositivo principal. Para comunicarse con los dispositivos de cliente en los componentes de Greengrass, configure el componente puente de MQTT para que haga lo siguiente:

- Reenviar mensajes MQTT de dispositivos de cliente a la publicación/suscripción local.
- Retransmita mensajes MQTT desde los dispositivos locales publish/subscribe a los clientes.

También puede interactuar con las sombras de dispositivo de cliente en los componentes de Greengrass. Para obtener más información, consulte Interacción y sincronización con las sombras de dispositivo de cliente.

Temas

- Configuración e implementación del componente puente de MQTT
- Recepción de mensajes MQTT desde los dispositivos de cliente
- Envío de mensajes MQTT a dispositivos de cliente

Configuración e implementación del componente puente de MQTT

El componente puente de MQTT utiliza una lista de asignaciones de temas, cada una de las cuales especifica un origen y un destino del mensaje. Para comunicarse con los dispositivos de cliente,

implemente el componente puente de MQTT y especifique cada tema de origen y destino en la configuración del componente.

Para implementar el componente puente MQTT en un dispositivo principal o en un grupo de dispositivos principales, cree una implementación que incluya el componente `aws.greengrass.clientdevices.mqtt.Bridge`. Especifique las asignaciones de temas, `mqttTopicMapping`, en la configuración del componente puente de MQTT en la implementación.

El siguiente ejemplo define una implementación que configura el componente de puente MQTT para retransmitir el `clients/MyClientDevice1/hello/world` tema desde los dispositivos cliente al intermediario local. publish/subscribe La actualización de configuración `merge` requiere un objeto JSON serializado. Para obtener más información, consulte Actualización de las configuraciones de los componentes.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldPubsub": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\"HelloWorldPubsub\":{\"topic\":\"clients/
MyClientDevice1/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"Pubsub\"}}}"
      }
    }
    ...
  }
}
```

Puede utilizar caracteres comodín de temas MQTT para reenviar mensajes sobre temas que coincidan con un filtro de temas. Si utiliza un puente de MQTT versión 2.2.0 o una versión posterior, puede usar comodines de temas MQTT en los filtros de temas cuando el agente de origen sea una publicación/suscripción local. Para obtener más información, consulte la configuración del componente puente de MQTT.

Recepción de mensajes MQTT desde los dispositivos de cliente

Puede suscribirse a los publish/subscribe temas locales que configure para el componente de puente de MQTT para recibir mensajes de los dispositivos cliente.

Cómo recibir mensajes MQTT desde dispositivos de cliente en componentes personalizados

1. Configure e implemente el componente bridge de MQTT para retransmitir los mensajes de un tema de MQTT donde los dispositivos cliente publican en un tema local. publish/subscribe
2. Utilice la interfaz publish/subscribe IPC local para suscribirse al tema en el que el puente MQTT transmite los mensajes. Para obtener más información, consulte Publicar/suscribir mensajes locales y SubscribeToTopic.

El tutorial Conectar y probar dispositivos de cliente incluye una sección en la que se desarrolla un componente que se suscribe a los mensajes de un dispositivo de cliente. Para obtener más información, consulte Paso 4: Desarrollar un componente que se comunique con los dispositivos de cliente.

Envío de mensajes MQTT a dispositivos de cliente

Puede publicar en los publish/subscribe temas locales que configure para el componente MQTT bridge a fin de enviar mensajes a los dispositivos cliente.

Cómo publicar mensajes MQTT en dispositivos de cliente en componentes personalizados

1. Configure e implemente el componente de puente de MQTT para retransmitir mensajes de un publish/subscribe tema local a un tema de MQTT al que se suscriban los dispositivos cliente.
2. Utilice la interfaz publish/subscribe IPC local para publicar en el tema en el que el puente MQTT transmite los mensajes. Para obtener más información, consulte Publicar/suscribir mensajes locales y PublishToTopic.

Interacción y sincronización con las sombras de dispositivo de cliente

Puede usar el componente administrador de sombras para administrar las sombras locales, incluidas las sombras de dispositivo de cliente. Puede usar el administrador de sombras para hacer lo siguiente:

- Interactuar con las sombras de dispositivo de cliente en los componentes de Greengrass.
- Sincronice las sombras de los dispositivos cliente con. AWS IoT Core

> **Note**
>
> El componente administrador de sombras no sincroniza las sombras con ellas de forma AWS IoT Core predeterminada. Debe configurar el componente administrador de sombras para especificar qué sombras de dispositivo de cliente desea sincronizar.

Temas

- Requisitos previos
- Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente
- Interacción con las sombras de dispositivo de cliente en los componentes
- Sincronice las sombras de los dispositivos cliente con AWS IoT Core

Requisitos previos

Para interactuar con las sombras de los dispositivos cliente y sincronizarlas con las sombras de los dispositivos cliente AWS IoT Core, un dispositivo básico debe cumplir los siguientes requisitos:

- El dispositivo principal debe ejecutar los siguientes componentes, además de los componentes de Greengrass para la compatibilidad con los dispositivos de cliente:
  - Núcleo de Greengrass versión 2.6.0 o posterior
  - Administrador de sombras versión 2.2.0 o posterior
  - Puente MQTT versión 2.2.0 o posterior

- El componente de autenticación del dispositivo de cliente debe configurarse para permitir que los dispositivos de cliente se comuniquen sobre temas de sombras de dispositivo.

Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente

De forma predeterminada, el componente administrador de sombras no administra las sombras de dispositivo de cliente. Para habilitar esta característica, debe retransmitir los mensajes MQTT entre los dispositivos de cliente y el componente administrador de sombras. Los dispositivos de cliente utilizan los mensajes MQTT para recibir y enviar actualizaciones de sombra de dispositivo. El componente administrador de sombras se suscribe a la interfaz local de publish/subscribe Greengrass, por lo que puede configurar el componente puente MQTT para retransmitir mensajes MQTT sobre temas ocultos de dispositivos.

El componente puente de MQTT utiliza una lista de asignaciones de temas, cada una de las cuales especifica un origen y un destino del mensaje. Para permitir que el componente administrador de sombras administre las sombras de dispositivo de cliente, implemente el componente puente de MQTT y especifique los temas de sombra para las sombras de dispositivo de cliente. Debe configurar el puente para retransmitir mensajes en ambas direcciones entre el MQTT local y el servicio de publicación/suscripción local.

Para implementar el componente puente MQTT en un dispositivo principal o en un grupo de dispositivos principales, cree una implementación que incluya el componente `aws.greengrass.clientdevices.mqtt.Bridge`. Especifique las asignaciones de temas, `mqttTopicMapping`, en la configuración del componente puente de MQTT en la implementación.

Utilice los siguientes ejemplos para configurar el componente puente d MQTT para permitir la comunicación entre los dispositivos de cliente y el componente administrador de sombras.

> **Note**
>
> Puede utilizar estos ejemplos de configuración en la consola. AWS IoT Greengrass Si utilizas la AWS IoT Greengrass API, la actualización de `merge` configuración requiere un objeto JSON serializado, por lo que debes serializar los siguientes objetos JSON en cadenas. Para obtener más información, consulte Actualización de las configuraciones de los componentes.

Example Ejemplo: administrar todas las sombras de dispositivo de cliente

El siguiente ejemplo de configuración de puente de MQTT permite que el administrador de sombras administre todas las sombras de todos los dispositivos de cliente.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Ejemplo: administrar las sombras de un dispositivo de cliente

El siguiente ejemplo de configuración de puente de MQTT permite que el administrador de sombras administre todas las sombras de un dispositivo de cliente llamado `MyClientDevice`.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Ejemplo: administrar una sombra con nombre para todos los dispositivos de cliente

El siguiente ejemplo de configuración de puente de MQTT permite que el administrador de sombras administre una sombra con nombre `DeviceConfiguration` para todos los dispositivos de cliente.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Ejemplo: administrar todas las sombras sin nombre de los dispositivos de cliente

El siguiente ejemplo de configuración de puente de MQTT permite que el administrador de sombras administre todas las sombras sin nombre, pero no las que tengan nombre, de todos los dispositivos de cliente.

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/delete/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
```

```
    },
    "GetShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/get/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "UpdateShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/update",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "UpdateShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/update/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Interacción con las sombras de dispositivo de cliente en los componentes

Puede desarrollar componentes personalizados que utilicen el servicio de sombra local para leer y modificar los documentos de sombra local de los dispositivos de cliente. Para obtener más información, consulte Interacción con las sombras de los componentes.

Sincronice las sombras de los dispositivos cliente con AWS IoT Core

Puede configurar el componente de administrador de sombras para sincronizar los estados ocultos de los dispositivos cliente locales con AWS IoT Core ellos. Para obtener más información, consulte Sincronice las sombras de los dispositivos locales con AWS IoT Core.

IPv6 Úselo para mensajería local

Puede configurar el componente detector de IP IPv6 para utilizarlo en el envío de mensajes locales.

> **Note**
>
> Debe tener un detector de IP v2.2.0 o posterior para poder usarlo IPv6 para enviar mensajes locales.

Puede implementar el componente detector de IP para detectar y usar IPv6 direcciones. Debe actualizar la configuración del componente del detector de IP para utilizarlo IPv6 en lugar de IPv4. Para obtener más información, consulte Uso del detector de IP para administrar automáticamente los puntos de conexión.

Temas

- Configure el detector IP para usar IPv6

Configure el detector IP para usar IPv6

Si tiene una configuración de red sencilla, como los dispositivos cliente en la misma red que el dispositivo principal, puede implementar el componente detector de IP IPv6 para usarlo en la mensajería local.

El componente detector de IP sobrescribe los puntos de conexión que se configuran manualmente.

> **Important**
>
> La AWS IoT política del dispositivo principal debe permitir el `greengrass:UpdateConnectivityInfo` permiso para usar el componente detector de IP. Para obtener más información, consulte AWS IoT políticas para las operaciones del plano de datos y Configure la política de AWS IoT cosas.

Puede realizar uno de los siguientes procedimientos para implementar el componente detector de IP:

- Utilice la página Configurar la detección en la consola. Para obtener más información, consulte Configuración de la detección en la nube (consola).
- Cree y revise las implementaciones para incluir el detector de IP. Puede usar la consola o la AWS API para administrar las implementaciones. AWS CLI Para obtener más información, consulte Crear implementaciones.

Implementación del componente detector de IP (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, elija la pestaña Componentes públicos y, luego, elija aws.greengrass.clientdevices.IPDetector.

3. En la página aws.greengrass.clientdevices.IPDetector, elija Implementar.
4. En Agregar a la implementación, elija una implementación existente para revisarla o cree una nueva y, a continuación, elija Siguiente.
5. Si opta por crear una nueva implementación, elija el dispositivo principal o el grupo de objetos de destino para la implementación. En la página Especificar el destino, en Destino de la implementación, elija un dispositivo principal o un grupo de objetos y, a continuación, elija Siguiente.
6. En la página Seleccionar componentes, compruebe que el componente aws.greengrass.clientdevices.IPDetector esté seleccionado y elija Siguiente.
7. En la página Configurar componentes, seleccione aws.greengrass.clientdevices.IPDetector y haga lo siguiente:

   a. Seleccione Configurar componente.

   b. En el cuadro Configurar aws.greengrass.clientdevices.IPDetector, en Actualizar configuración, en Configuración para combinar, puede introducir una actualización de configuración para configurar el componente detector de IP. Puede especificar una de las siguientes opciones de configuración. Establezca `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true`. A continuación, puede actualizar las demás opciones IPv6 de configuración.

      - `defaultPort`: (opcional) el puerto del agente MQTT para informar cuando este componente detecta direcciones IP. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883.
      - `includeIPv4LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar y notificar las direcciones de IPv4 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.
      - `includeIPv4LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.
      - `includeIPv6LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv6 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice

esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.

- `includeIPv6LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv6 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv4Addrs`: (opcional) el valor predeterminado está establecido en verdadero. Puede activar esta opción para publicar IPv4 las direcciones que se encuentran en el dispositivo principal. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv6Addrs`— (Opcional) Puede activar esta opción para publicar IPv6 las direcciones que se encuentran en el dispositivo principal. Configure `includeIPv4Addrs` en `false` para usar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.

La actualización de la configuración podría parecerse al siguiente ejemplo.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false,
  "includeIPv6LoopbackAddrs": true,
  "includeIPv6LinkLocalAddrs": true,
  "includeIPv4Addrs": false,
  "includeIPv6Addrs": true
}
```

c. Elija Confirmar para cerrar el cuadro y, a continuación, elija Siguiente.

8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Revisar, elija Implementar.

   La implementación puede tardar hasta un minuto para completarse.

Implementación del componente detector de IP (AWS CLI)

Para implementar el componente detector de IP, cree un documento de implementación que incluya `aws.greengrass.clientdevices.IPDetector` en el objeto `components` y especifique la actualización de configuración del componente. Siga las instrucciones en Crear implementaciones para crear una implementación nueva o revisar una implementación existente.

Puede especificar cualquiera de las siguientes opciones para configurar el componente detector de IP cuando cree el documento de implementación:

- `defaultPort`: (opcional) el puerto del agente MQTT para informar cuando este componente detecta direcciones IP. Debe especificar este parámetro si configura el agente MQTT para que utilice un puerto diferente al puerto predeterminado 8883.
- `includeIPv4LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv4 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema.
- `includeIPv4LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv4 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática.
- `includeIPv6LoopbackAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones de IPv6 bucle invertido. Estas son direcciones IP, por ejemplo, `localhost`, donde un dispositivo puede comunicarse consigo mismo. Utilice esta opción en entornos de prueba en los que el dispositivo principal y el dispositivo de cliente se ejecuten en el mismo sistema. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv6LinkLocalAddrs`— (Opcional) Puede activar esta opción para detectar e informar sobre las direcciones locales de los IPv6 enlaces. Utilice esta opción si la red del dispositivo principal no tiene el Protocolo de configuración dinámica de host (DHCP) ni direcciones IP asignadas de forma estática. Debe configurar `includeIPv4Addrs` en `false` y `includeIPv6Addrs` en `true` para utilizar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.
- `includeIPv4Addrs`: (opcional) el valor predeterminado está establecido en verdadero. Puede activar esta opción para publicar IPv4 las direcciones que se encuentran en el dispositivo principal. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.

- `includeIPv6Addrs`— (Opcional) Puede activar esta opción para publicar IPv6 las direcciones que se encuentran en el dispositivo principal. Configure `includeIPv4Addrs` en `false` para usar esta opción. Debe tener el detector de IP versión 2.2.0 o posterior para usar esta opción.

En el siguiente ejemplo de documento de despliegue parcial se especifica el uso IPv6.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Solución de problemas de los dispositivos de cliente

Utilice la información de solución de problemas y las soluciones de esta sección para resolver problemas con los dispositivos de cliente de Greengrass y los componentes de los dispositivos de cliente.

Temas

- Problemas de detección de Greengrass
- Problemas de conexión con MQTT

Problemas de detección de Greengrass

Utilice la siguiente información como ayuda para solucionar problemas con la detección de Greengrass. Estos problemas pueden producirse cuando los dispositivos de cliente utilizan la API de detección de Greengrass para identificar un dispositivo principal de Greengrass al que se pueden conectar.

Temas

- Problemas de detección de Greengrass (API HTTP)

- Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para Python)
- Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para C++)
- Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para) JavaScript
- Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para Java)

Problemas de detección de Greengrass (API HTTP)

Utilice la siguiente información como ayuda para solucionar problemas con la detección de Greengrass. Es posible que vea estos errores si prueba la API de detección con cURL.

Temas

- curl: (52) Empty reply from server
- HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}
- HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

curl: (52) Empty reply from server

Es posible que aparezca este error si especifica un AWS IoT certificado inactivo en la solicitud.

Compruebe que el dispositivo de cliente tiene un certificado adjunto y que el certificado está activo. Para obtener más información, consulte Adjuntar un objeto o una política a un certificado de cliente y Activar o desactivar un certificado de cliente en la Guía para desarrolladores de AWS IoT Core .

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

Es posible que aparezca este error si el dispositivo de cliente no tiene permiso para realizar llamadas `greengrass:Discover` por sí mismo.

Compruebe que el certificado del dispositivo de cliente tenga una política que permita `greengrass:Discover`. No puede usar variables de política de objetos (`iot:Connection.Thing.*`) en la sección `Resource` correspondiente a este permiso. Para obtener más información, consulte Autenticación y autorización de detección.

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

En estos casos, aparece el siguiente error:

- El dispositivo cliente no está asociado a ningún dispositivo o AWS IoT Greengrass V1 grupo principal de Greengrass.
- Ninguno de los dispositivos o AWS IoT Greengrass V1 grupos principales de Greengrass asociados al dispositivo cliente tiene un punto final intermediario MQTT.
- Ninguno de los dispositivos principales de Greengrass asociados al dispositivo de cliente ejecuta el componente de autenticación del dispositivo de cliente.

Compruebe que el dispositivo de cliente esté asociado al dispositivo principal al que desea que se conecte. A continuación, compruebe que el dispositivo principal ejecute el componente de autenticación del dispositivo de cliente y que tenga al menos un punto de conexión de agente MQTT. Para obtener más información, consulte los siguientes temas:

- Asociación de los dispositivos de cliente
- Administración de puntos de conexión del dispositivo principal
- Configuración de la detección en la nube (consola)

Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para Python)

Utilice la siguiente información como ayuda para solucionar problemas con la detección de Greengrass en SDK para dispositivos con AWS IoT v2 para Python.

Temas

- awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

Es posible que aparezca este error si especifica un AWS IoT certificado inactivo en la solicitud.

Compruebe que el dispositivo de cliente tiene un certificado adjunto y que el certificado está activo. Para obtener más información, consulte Adjuntar un objeto o una política a un certificado de cliente y Activar o desactivar un certificado de cliente en la Guía para desarrolladores de AWS IoT Core .

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

Es posible que aparezca este error si el dispositivo de cliente no tiene permiso para realizar llamadas `greengrass:Discover` por sí mismo.

Compruebe que el certificado del dispositivo de cliente tenga una política que permita `greengrass:Discover`. No puede usar variables de política de objetos (`iot:Connection.Thing.*`) en la sección `Resource` correspondiente a este permiso. Para obtener más información, consulte Autenticación y autorización de detección.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

En estos casos, aparece el siguiente error:

- El dispositivo cliente no está asociado a ningún dispositivo o AWS IoT Greengrass V1 grupo principal de Greengrass.
- Ninguno de los dispositivos o AWS IoT Greengrass V1 grupos principales de Greengrass asociados al dispositivo cliente tiene un punto final intermediario MQTT.
- Ninguno de los dispositivos principales de Greengrass asociados al dispositivo de cliente ejecuta el componente de autenticación del dispositivo de cliente.

Compruebe que el dispositivo de cliente esté asociado al dispositivo principal al que desea que se conecte. A continuación, compruebe que el dispositivo principal ejecute el componente de autenticación del dispositivo de cliente y que tenga al menos un punto de conexión de agente MQTT. Para obtener más información, consulte los siguientes temas:

- Asociación de los dispositivos de cliente
- Administración de puntos de conexión del dispositivo principal

- Configuración de la detección en la nube (consola)

Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para C++)

Utilice la siguiente información como ayuda para solucionar problemas con la detección de Greengrass en SDK para dispositivos con AWS IoT v2 para C++.

Temas

- aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Es posible que aparezca este error si especifica un AWS IoT certificado inactivo en la solicitud.

Compruebe que el dispositivo de cliente tiene un certificado adjunto y que el certificado está activo. Para obtener más información, consulte Adjuntar un objeto o una política a un certificado de cliente y Activar o desactivar un certificado de cliente en la Guía para desarrolladores de AWS IoT Core .

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

Es posible que aparezca este error si el dispositivo de cliente no tiene permiso para realizar llamadas `greengrass:Discover` por sí mismo.

Compruebe que el certificado del dispositivo de cliente tenga una política que permita `greengrass:Discover`. No puede usar variables de política de objetos (`iot:Connection.Thing.*`) en la sección `Resource` correspondiente a este permiso. Para obtener más información, consulte Autenticación y autorización de detección.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

En estos casos, aparece el siguiente error:

- El dispositivo cliente no está asociado a ningún dispositivo o AWS IoT Greengrass V1 grupo principal de Greengrass.

- Ninguno de los dispositivos o AWS IoT Greengrass V1 grupos principales de Greengrass asociados al dispositivo cliente tiene un punto final intermediario MQTT.
- Ninguno de los dispositivos principales de Greengrass asociados al dispositivo de cliente ejecuta el componente de autenticación del dispositivo de cliente.

Compruebe que el dispositivo de cliente esté asociado al dispositivo principal al que desea que se conecte. A continuación, compruebe que el dispositivo principal ejecute el componente de autenticación del dispositivo de cliente y que tenga al menos un punto de conexión de agente MQTT. Para obtener más información, consulte los siguientes temas:

- Asociación de los dispositivos de cliente
- Administración de puntos de conexión del dispositivo principal
- Configuración de la detección en la nube (consola)

Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para) JavaScript

Utilice la siguiente información para solucionar problemas relacionados con el descubrimiento de Greengrass en SDK para dispositivos con AWS IoT la versión 2 para. JavaScript

Temas

- Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- Error: Discovery failed (headers: [object Object]) { response_code: 403 }
- Error: Discovery failed (headers: [object Object]) { response_code: 404 }
- Error: Discovery failed (headers: [object Object])

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Es posible que aparezca este error si especifica un AWS IoT certificado inactivo en la solicitud.

Compruebe que el dispositivo de cliente tiene un certificado adjunto y que el certificado está activo. Para obtener más información, consulte Adjuntar un objeto o una política a un certificado de cliente y Activar o desactivar un certificado de cliente en la Guía para desarrolladores de AWS IoT Core .

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

Es posible que aparezca este error si el dispositivo de cliente no tiene permiso para realizar llamadas `greengrass:Discover` por sí mismo.

Compruebe que el certificado del dispositivo de cliente tenga una política que permita `greengrass:Discover`. No puede usar variables de política de objetos (`iot:Connection.Thing.*`) en la sección `Resource` correspondiente a este permiso. Para obtener más información, consulte Autenticación y autorización de detección.

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

En estos casos, aparece el siguiente error:

- El dispositivo cliente no está asociado a ningún dispositivo o AWS IoT Greengrass V1 grupo principal de Greengrass.
- Ninguno de los dispositivos o AWS IoT Greengrass V1 grupos principales de Greengrass asociados al dispositivo cliente tiene un punto final intermediario MQTT.
- Ninguno de los dispositivos principales de Greengrass asociados al dispositivo de cliente ejecuta el componente de autenticación del dispositivo de cliente.

Compruebe que el dispositivo de cliente esté asociado al dispositivo principal al que desea que se conecte. A continuación, compruebe que el dispositivo principal ejecute el componente de autenticación del dispositivo de cliente y que tenga al menos un punto de conexión de agente MQTT. Para obtener más información, consulte los siguientes temas:

- Asociación de los dispositivos de cliente
- Administración de puntos de conexión del dispositivo principal
- Configuración de la detección en la nube (consola)

Error: Discovery failed (headers: [object Object])

Es posible que vea este error (sin un código de respuesta HTTP) al ejecutar el ejemplo de detección de Greengrass. Este error puede producirse por varios motivos.

- Es posible que aparezca este error si el dispositivo de cliente no tiene permiso para realizar llamadas `greengrass:Discover` por sí mismo.

Compruebe que el certificado del dispositivo de cliente tenga una política que permita `greengrass:Discover`. No puede usar variables de política de objetos (`iot:Connection.Thing.*`) en la sección `Resource` correspondiente a este permiso. Para obtener más información, consulte Autenticación y autorización de detección.

- En estos casos, aparece el siguiente error:
  - El dispositivo cliente no está asociado a ningún dispositivo o AWS IoT Greengrass V1 grupo principal de Greengrass.
  - Ninguno de los dispositivos o AWS IoT Greengrass V1 grupos principales de Greengrass asociados al dispositivo cliente tiene un punto final intermediario MQTT.
  - Ninguno de los dispositivos principales de Greengrass asociados al dispositivo de cliente ejecuta el componente de autenticación del dispositivo de cliente.

  Compruebe que el dispositivo de cliente esté asociado al dispositivo principal al que desea que se conecte. A continuación, compruebe que el dispositivo principal ejecute el componente de autenticación del dispositivo de cliente y que tenga al menos un punto de conexión de agente MQTT. Para obtener más información, consulte los siguientes temas:
  - Asociación de los dispositivos de cliente
  - Administración de puntos de conexión del dispositivo principal
  - Configuración de la detección en la nube (consola)

Problemas de descubrimiento de Greengrass (SDK para dispositivos con AWS IoT versión 2 para Java)

Utilice la siguiente información como ayuda para solucionar problemas con la detección de Greengrass en SDK para dispositivos con AWS IoT v2 para Java.

Temas

- software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)
- java.lang.RuntimeException: Error x-amzn-ErrorType(403)
- java.lang.RuntimeException: Error x-amzn-ErrorType(404)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

Es posible que aparezca este error si especifica un AWS IoT certificado inactivo en la solicitud.

Compruebe que el dispositivo de cliente tiene un certificado adjunto y que el certificado está activo. Para obtener más información, consulte Adjuntar un objeto o una política a un certificado de cliente y Activar o desactivar un certificado de cliente en la Guía para desarrolladores de AWS IoT Core .

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

Es posible que aparezca este error si el dispositivo de cliente no tiene permiso para realizar llamadas `greengrass:Discover` por sí mismo.

Compruebe que el certificado del dispositivo de cliente tenga una política que permita `greengrass:Discover`. No puede usar variables de política de objetos (`iot:Connection.Thing.*`) en la sección `Resource` correspondiente a este permiso. Para obtener más información, consulte Autenticación y autorización de detección.

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

En estos casos, aparece el siguiente error:

- El dispositivo cliente no está asociado a ningún dispositivo o AWS IoT Greengrass V1 grupo principal de Greengrass.
- Ninguno de los dispositivos o AWS IoT Greengrass V1 grupos principales de Greengrass asociados al dispositivo cliente tiene un punto final intermediario MQTT.
- Ninguno de los dispositivos principales de Greengrass asociados al dispositivo de cliente ejecuta el componente de autenticación del dispositivo de cliente.

Compruebe que el dispositivo de cliente esté asociado al dispositivo principal al que desea que se conecte. A continuación, compruebe que el dispositivo principal ejecute el componente de autenticación del dispositivo de cliente y que tenga al menos un punto de conexión de agente MQTT. Para obtener más información, consulte los siguientes temas:

- Asociación de los dispositivos de cliente
- Administración de puntos de conexión del dispositivo principal
- Configuración de la detección en la nube (consola)

Problemas de conexión con MQTT

Utilice la siguiente información para solucionar problemas con las conexiones MQTT del dispositivo de cliente. Estos problemas pueden producirse cuando los dispositivos de cliente intentan conectarse a un dispositivo principal a través de MQTT.

Temas

- io.moquette.broker.Authorizator: Client does not have read permissions on the topic
- Problemas de conexión con MQTT (Python)
- Problemas de conexión con MQTT (C++)
- Problemas de conexión con MQTT (Java)
- Problemas de conexión con MQTT () JavaScript

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

Es posible que vea este error en los registros de Greengrass cuando un dispositivo de cliente intenta suscribirse a un tema de MQTT para el que no tiene permiso. El mensaje de error incluye el tema.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:subscribe` para el tema.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

Problemas de conexión con MQTT (Python)

Utilice la siguiente información para solucionar problemas con las conexiones MQTT del dispositivo de cliente cuando utilice SDK para dispositivos con AWS IoT versión 2 para Python.

Temas

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

Problemas de conexión con MQTT (C++)

Utilice la siguiente información para solucionar problemas con las conexiones MQTT del dispositivo de cliente cuando utilice SDK para dispositivos con AWS IoT versión 2 para C++.

Temas

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

Problemas de conexión con MQTT (Java)

Utilice la siguiente información para solucionar problemas con las conexiones MQTT del dispositivo de cliente cuando utilice SDK para dispositivos con AWS IoT versión 2 para Java.

Temas

- software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

Problemas de conexión con MQTT () JavaScript

Utilice la siguiente información para solucionar problemas con las conexiones MQTT de los dispositivos cliente cuando utilice la SDK para dispositivos con AWS IoT versión 2 para. JavaScript

Temas

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Es posible que aparezca este error si el componente de autenticación del dispositivo de cliente no define una política de autorización del dispositivo de cliente que le conceda permiso para conectarse.

Compruebe que la configuración del componente de autenticación del dispositivo de cliente incluya lo siguiente:

- Un grupo de dispositivos que coincida con el dispositivo de cliente.
- Una política de autorización de dispositivos de cliente para ese grupo de dispositivos que otorga el permiso `mqtt:connect` para el dispositivo de cliente.

Para obtener más información acerca de cómo implementar y configurar el componente de autenticación del dispositivo de cliente, consulte lo siguiente:

- Configuración de la detección en la nube (consola)
- Autenticación del dispositivo de cliente
- Crear implementaciones

Interacción con las sombras de dispositivo

Los dispositivos principales de Greengrass pueden interactuar con las sombras de dispositivo de AWS IoT mediante componentes. Una sombra es un documento JSON que se usa para almacenar la información del estado actual o deseado de un objeto AWS IoT. Las sombras pueden hacer que el estado de un dispositivo esté disponible para otros componentes de AWS IoT Greengrass, ya sea que el dispositivo esté conectado a AWS IoT o no. Cada dispositivo AWS IoT tiene su propia sombra clásica sin nombre. También puede crear varias sombras con nombre para cada dispositivo.

Los dispositivos y servicios pueden crear, actualizar y eliminar sombras en la nube mediante MQTT y los temas de sombra MQTT reservados, HTTP mediante la API de REST de sombra de dispositivo y la AWS CLI para AWS IoT.

El componente administrador de sombras permite a sus componentes de Greengrass crear, actualizar y eliminar sombras locales mediante el servicio de sombras local y los temas de sombra local de publicación/suscripción. El administrador de sombras también administra el almacenamiento de estos documentos de sombras locales en su dispositivo principal y gestiona la sincronización de la información del estado de las sombras con las sombras en la nube.

También puede utilizar el componente administrador de sombras para administrar las sombras locales de los dispositivos de cliente que se conectan al dispositivo principal. Para permitir que el administrador de sombras administre las sombras de dispositivo de cliente, debe configurar el componente de puente de MQTT para reenviar mensajes entre el agente MQTT local y el servicio de publicación/suscripción local. Para obtener más información, consulte Interacción y sincronización con las sombras de dispositivo de cliente.

Para obtener más información acerca de los conceptos de sombra de dispositivo de AWS IoT, consulte Servicio de sombra de dispositivo AWS IoT en la Guía para desarrolladores de AWS IoT.

Temas

- Interacción con las sombras de los componentes
- Sincronice las sombras de los dispositivos locales con AWS IoT Core

Interacción con las sombras de los componentes

Puede desarrollar componentes personalizados, incluidos los componentes de la función de Lambda, que utilicen el servicio de sombra local para leer y modificar documentos de sombra locales y documentos de sombra de dispositivo de cliente.

Los componentes personalizados interactúan con el servicio oculto local mediante las bibliotecas AWS IoT Greengrass Core IPC del SDK para dispositivos con AWS IoT. El componente administrador de sombras habilita el servicio de sombra local en su dispositivo principal.

Para implementar el componente de administrador de sombras en un dispositivo principal de Greengrass, cree una implementación que incluya el componente `aws.greengrass.ShadowManager`.

> **Note**
>
> De forma predeterminada, la implementación del componente administrador de sombras solo permite las operaciones de sombras locales. AWS IoT Greengrass Para poder sincronizar la información sobre el estado de las sombras de los dispositivos principales o de cualquier sombra de los dispositivos cliente con los documentos de sombra de nube correspondientes AWS IoT Core, debe crear una actualización de configuración para el componente de administrador de sombras que incluya el `synchronize` parámetro. Para obtener más información, consulte Sincronice las sombras de los dispositivos locales con AWS IoT Core.

Temas

- Recuperación y modificación de los estados de sombra
- Reacción a los cambios en el estado de sombra

Recuperación y modificación de los estados de sombra

Las operaciones de IPC de sombra recuperan y actualizan la información sobre el estado de los documentos de sombra local. El componente administrador de sombras se encarga del almacenamiento de estos documentos de sombra en el dispositivo principal.

Cómo modificar los estados de sombra local

1. Agregue políticas de autorización a la receta de su componente personalizado para permitir que el componente reciba mensajes sobre temas de sombra local.

   Para ver ejemplos de políticas de autorización, consulte Ejemplos de políticas de autorización de IPC de sombra local.
2. Utilice las operaciones de IPC de sombras para recuperar y modificar la información sobre el estado de sombras. Para obtener más información sobre el uso de operaciones de IPC de sombras en el código de los componentes, consulte Interactúe con las sombras locales.

> **Note**
>
> Para permitir que un dispositivo principal interactúe con las sombras de dispositivos de cliente, también debe configurar e implementar el componente puente MQTT. Para obtener más información, consulte Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente.

Reacción a los cambios en el estado de sombra

Los componentes de Greengrass utilizan la publish/subscribe interfaz local para comunicarse en un dispositivo central. Para permitir que un componente personalizado reaccione ante los cambios en el estado de sombra, puede suscribirse a los publish/subscribe temas locales. Esto permite que el componente reciba mensajes sobre los temas de sombra local y, a continuación, actúe en función de esos mensajes.

Los temas ocultos locales utilizan el mismo formato que los temas MQTT ocultos de AWS IoT dispositivos. Para obtener más información sobre temas de sombras, consulte Temas MQTT de sombra de dispositivo en la Guía para desarrolladores de AWS IoT .

Reacción a los cambios en el estado de sombra local

1. Agregue políticas de control de acceso a la receta de su componente personalizado para permitir que el componente reciba mensajes sobre temas de sombra local.

   Para ver ejemplos de políticas de autorización, consulte Ejemplos de políticas de autorización de IPC de sombra local.

2. Para iniciar una acción personalizada en un componente, utilice las operaciones de IPC `SubscribeToTopic` para suscribirse a los temas de sombra sobre los que desee recibir mensajes. Para obtener más información sobre el uso de operaciones de publish/subscribe IPC locales en el código de los componentes, consulte. Publicar/suscribir mensajes locales
3. Para invocar una función Lambda, utilice la configuración de la fuente de eventos para proporcionar el nombre del tema paralelo y especificar que se trata de un publish/subscribe tema local. Para obtener más información sobre la creación de componentes de la función de Lambda, consulte Ejecución de funciones de AWS Lambda.

> **Note**
>
> Para permitir que un dispositivo principal interactúe con las sombras de dispositivos de cliente, también debe configurar e implementar el componente puente MQTT. Para obtener más información, consulte Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente.

Sincronice las sombras de los dispositivos locales con AWS IoT Core

El componente administrador de sombras permite sincronizar AWS IoT Greengrass los estados de sombra de los dispositivos locales con AWS IoT Core. Debe modificar la configuración del componente administrador de sombras para incluir el parámetro de `synchronization` configuración y AWS IoT especificar los nombres de los dispositivos y las sombras que desea sincronizar.

Al configurar el administrador de sombras para que sincronice las sombras, sincroniza todos los cambios de estado de las sombras especificadas, independientemente de si los cambios se producen en documentos de sombra locales o en documentos de sombra en la nube.

También puede especificar si el componente administrador de sombras sincroniza las sombras en tiempo real o en intervalos periódicos. De forma predeterminada, el componente administrador de sombras sincroniza las sombras en tiempo real, por lo que el dispositivo principal envía y recibe actualizaciones ocultas desde y hacia el AWS IoT Core momento en que se produce cada actualización. Puede configurar intervalos periódicos para reducir el uso de ancho de banda y los cargos.

Temas

- Requisitos previos
- Configuración del componente administrador de sombras
- Sincronización de sombras locales
- Comportamiento conflictivo en la combinación de sombras

Requisitos previos

Para sincronizar las sombras locales AWS IoT Core, debe configurar la AWS IoT política del dispositivo principal de Greengrass para permitir las siguientes acciones de política AWS IoT Core clandestina.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Para obtener más información, consulte los siguientes temas:

- AWS IoT Core acciones políticas en la Guía AWS IoT para desarrolladores
- AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales
- Actualice la AWS IoT política de un dispositivo principal

Configuración del componente administrador de sombras

El administrador de sombras necesita una lista de asignaciones de nombres de sombras, con el fin de sincronizar la información sobre el estado de las sombras en los documentos de sombras locales con los documentos de sombras en la nube en AWS IoT Core.

Para sincronizar los estados de sombra, cree una implementación que incluya el componente `aws.greengrass.ShadowManager` y especifique las sombras que desea sincronizar en el parámetro de configuración `synchronize` de la configuración del administrador de sombras en la implementación.

> **Note**
>
> Para permitir que un dispositivo principal interactúe con las sombras de dispositivos de cliente, también debe configurar e implementar el componente puente MQTT. Para obtener más información, consulte Habilitación del administrador de sombras para que se comunique con los dispositivos de cliente.

El siguiente ejemplo de actualización de configuración indica al componente administrador de sombras que sincronice las siguientes sombras con AWS IoT Core:

- La sombra clásica para el dispositivo principal
- La `MyCoreShadow` con nombre del dispositivo principal
- La sombra clásica para un objeto del IoT llamada `MyDevice2`
- Las sombras con nombre `MyShadowA` y `MyShadowB` para un objeto del IoT llamada `MyDevice1`

Esta actualización de configuración especifica sincronizar las sombras con ellas AWS IoT Core en tiempo real. Si utiliza la versión 2.1.0 o posterior del administrador de sombras, puede configurar el componente administrador de sombras para que sincronice las sombras a intervalos periódicos. Para configurar esta característica, cambie la estrategia de sincronización a `periodic` y especifique un `delay` en segundos para el intervalo. Para obtener más información, consulte el parámetro de configuración de la estrategia del componente administrador de sombras.

Esta actualización de configuración especifica que las sombras se sincronicen en ambas direcciones entre AWS IoT Core y el dispositivo principal. Si utiliza la versión 2.2.0 o posterior del administrador de sombras, puede configurar el componente administrador de sombras para que sincronice las sombras en una dirección. Para configurar esta característica, cambie la sincronización `direction` a `deviceToCloud` o `cloudToDevice`. Para obtener más información, consulte el parámetro de configuración de la dirección del componente administrador de sombras.

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
```

```
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
}
```

Sincronización de sombras locales

Cuando el dispositivo principal de Greengrass está conectado a la AWS IoT nube, el administrador de sombras realiza las siguientes tareas para las sombras que especifique en la configuración del componente. El comportamiento depende de la opción de configuración especificada en la dirección de sincronización de sombras. De forma predeterminada, el administrador de sombras usa la opción `betweenDeviceAndCloud` para sincronizar las sombras en ambas direcciones. Si utiliza la versión 2.2.0 o posterior del administrador de sombras, puede configurar el dispositivo principal para que sincronice las sombras en una dirección, que puede ser `cloudToDevice` o `deviceToCloud`.

- Si la configuración de la dirección de sincronización de sombras es `betweenDeviceAndCloud` o `cloudToDevice`, el administrador de sombras recupera la información de estado notificada del documento de sombra en la nube en AWS IoT Core. A continuación, actualiza los documentos de sombras almacenados localmente para sincronizar el estado del dispositivo.
- Si la configuración de la dirección de sincronización de sombra es `betweenDeviceAndCloud` o`deviceToCloud`, el administrador de sombras publica el estado actual del dispositivo en el documento de sombra en la nube.

Comportamiento conflictivo en la combinación de sombras

En algunos casos, como cuando el dispositivo principal está desconectado de Internet, es posible que una sombra cambie en el servicio de sombra local y en la AWS IoT nube antes de que el administrador de sombras sincronice los cambios. Como resultado, los estados deseados y notificados difieren entre el servicio paralelo local y la nube AWS IoT

Cuando el administrador de sombras sincroniza la sombra, fusiona los cambios de acuerdo con el siguiente comportamiento:

- Si utiliza una versión del administrador de sombras anterior a la versión 2.2.0, o si especifica la dirección de sincronización de sombras `betweenDeviceAndCloud`, se aplica el siguiente comportamiento:
  - Cuando se produce un conflicto de fusión en el estado deseado de una sombra, el administrador de la sombra sobrescribe la sección conflictiva del documento alternativo local con el valor de la AWS IoT nube.
  - Cuando se produce un conflicto de fusión en el estado registrado de una sombra, el administrador de la sombra sobrescribe la sección conflictiva de la sombra de la AWS IoT nube con el valor del documento paralelo local.
- Al especificar la dirección de sincronización de las `deviceToCloud` sombras, el administrador de sombras sobrescribe la sección conflictiva de la sombra en la AWS IoT nube con el valor del documento paralelo local.
- Al especificar la dirección de sincronización de las `cloudToDevice` sombras, el administrador de sombras sobrescribe la sección conflictiva del documento paralelo local con el valor de la nube. AWS IoT

Administración de flujos de datos en los dispositivos principales de Greengrass

AWS IoT Greengrass stream manager hace que sea más eficiente y confiable transferir datos de IoT de gran volumen al Nube de AWS. Stream Manager procesa los flujos de datos en el AWS IoT Greengrass Core antes de exportarlos al Nube de AWS. Stream Manager se integra en escenarios periféricos comunes, como la inferencia del aprendizaje automático (ML), en la que el dispositivo AWS IoT Greengrass Core procesa y analiza los datos antes de exportarlos a los destinos de almacenamiento locales Nube de AWS o a los destinos de almacenamiento.

El administrador de flujos proporciona una interfaz común para simplificar el desarrollo de componentes personalizados para que no sea necesario crear una funcionalidad de administración de flujos personalizada. Sus componentes pueden usar un mecanismo estandarizado para procesar flujos de gran volumen y administrar las políticas locales de retención de datos. Puede definir políticas para el tipo de almacenamiento, el tamaño y la retención de datos según cada flujo para controlar cómo el administrador de flujos procesa y exporta datos.

El administrador de flujos trabaja en entornos con conectividad intermitente o limitada. Puede definir el uso del ancho de banda, el comportamiento de los tiempos de espera y la forma en que el AWS IoT Greengrass Core gestiona los datos de la transmisión cuando está conectado o desconectado. También puede establecer prioridades para controlar el orden en el que AWS IoT Greengrass Core exporta flujos a la Nube de AWS. Esto le permite gestionar los datos críticos antes que otros datos.

Puede configurar el administrador de transmisiones para que exporte automáticamente los datos a él Nube de AWS para su almacenamiento o posterior procesamiento y análisis. El administrador de flujos admite exportar a los siguientes destinos de la Nube de AWS :

- Canales de entrada AWS IoT Analytics. AWS IoT Analytics le permite realizar análisis avanzados de sus datos para ayudarle a tomar decisiones empresariales y mejorar los modelos de aprendizaje automático. Para obtener más información, consulte ¿Qué es AWS IoT Analytics? en la Guía del usuario de AWS IoT Analytics .
- Flujos de Amazon Kinesis Data Streams. Puede usar Kinesis Data Streams para agregar grandes volúmenes de datos y cargarlos en un almacén MapReduce de datos o un clúster. Para obtener más información, consulte Qué son los Amazon Kinesis Data Streams en la Guía para desarrolladores de Amazon Kinesis Data Streams.

- Propiedades de los activos en. AWS IoT SiteWise AWS IoT SiteWise le permite recopilar, organizar y analizar datos de equipos industriales a escala. Para obtener más información, consulte ¿Qué es AWS IoT SiteWise? en la Guía AWS IoT SiteWise del usuario.
- Objetos en Amazon Simple Storage Service Amazon S3. Puede utilizar Amazon S3 para almacenar y recuperar grandes cantidades de datos. Para obtener más información, consulte ¿Qué es Amazon S3? en la Guía para desarrolladores de Amazon Simple Storage Service.

Flujo de trabajo de la administración de secuencias

Sus aplicaciones de IoT interactúan con el administrador de flujos a través del SDK del administrador de flujos.

En un flujo de trabajo simple, un componente del AWS IoT Greengrass núcleo consume datos de IoT, como métricas de temperatura y presión de series temporales. El componente podría filtrar o comprimir los datos y luego llamar al SDK del administrador de flujos para escribir los datos en un flujo en el administrador de flujos. Stream Manager puede exportar la transmisión a la transmisión Nube de AWS automáticamente en función de las políticas que defina para la transmisión. Los componentes también pueden enviar datos directamente a bases de datos locales o repositorios de almacenamiento.

Sus aplicaciones de IoT pueden incluir múltiples componentes personalizados que leen o escriben en flujos. Estos componentes pueden leer y escribir en las transmisiones para filtrar, agregar y analizar los datos del dispositivo AWS IoT Greengrass principal. Esto permite responder rápidamente a los eventos locales y extraer información valiosa antes de que los datos se transfieran del núcleo a los Nube de AWS destinos locales.

Para empezar, implemente el componente administrador de transmisiones en su dispositivo AWS IoT Greengrass principal. En la implementación, configure los parámetros del componente del administrador de flujos para definir los ajustes que se apliquen a todos los flujos del dispositivo principal de Greengrass. Use estos parámetros para controlar cómo el administrador de flujos almacena, procesa y exporta flujos en función de las necesidades de su negocio y las restricciones del entorno.

Después de configurar el administrador de flujos, puede crear e implementar sus aplicaciones de IoT. Por lo general, se trata de componentes personalizados que utilizan `StreamManagerClient` en el SDK del administrador de flujos para crear flujos e interactuar con ellos. Cuando crea un flujo, puede definir las políticas por flujo, como los destinos de exportación, la prioridad y la persistencia.

Requisitos

Se aplican los siguientes requisitos para el administrador de flujos:

- Stream Manager requiere un mínimo de 70 MB de RAM además del software AWS IoT Greengrass Core. El requisito total de memoria depende de la carga de trabajo.
- AWS IoT Greengrass los componentes deben usar el SDK de Stream Manager para interactuar con Stream Manager. El SDK del administrador de flujos está disponible en los siguientes lenguajes:
  - SDK del administrador de flujos para Java (versión 1.1.0 o posterior)
  - SDK del administrador de flujos para Node.js (versión 1.1.0 o posterior)
  - SDK del administrador de flujos para Python (versión 1.1.0 o posterior)
- AWS IoT Greengrass los componentes deben especificar el componente del administrador de transmisiones (`aws.greengrass.StreamManager`) como una dependencia en su receta para usar el administrador de transmisiones.

  > **Note**
  >
  > Si usa el administrador de flujos para exportar datos a la nube, no puede actualizar la versión 2.0.7 del componente de administrador de flujos a una versión entre la 2.0.8 y la 2.0.11. Si implementa el administrador de flujos por primera vez, le recomendamos que implemente la última versión del componente administrador de flujos.

- Si define los destinos de Nube de AWS exportación para una transmisión, debe crear sus objetivos de exportación y conceder permisos de acceso en la función de dispositivo de Greengrass. Según el destino, es posible que también se apliquen otros requisitos. Para obtener más información, consulte lo siguiente:
  - the section called “AWS IoT Analytics canales”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “AWS IoT SiteWise propiedades de los activos”
  - the section called “Objetos de Amazon S3”

  Usted es responsable del mantenimiento de estos Nube de AWS recursos.

Seguridad de los datos

Cuando utilice el administrador de secuencias, tenga en cuenta las siguientes consideraciones de seguridad.

Seguridad de los datos locales

AWS IoT Greengrass no cifra los datos de la transmisión en reposo o en tránsito entre los componentes locales del dispositivo principal.

- Datos en reposo. Los datos de secuencias se almacenan localmente en un directorio de almacenamiento. Para garantizar la seguridad de los datos, AWS IoT Greengrass se basa en los permisos de los archivos y en el cifrado de disco completo, si está activado. Puede utilizar el parámetro opcional STREAM_MANAGER_STORE_ROOT_DIR para especificar el directorio de almacenamiento. Si cambia este parámetro más adelante para usar un directorio de almacenamiento diferente, AWS IoT Greengrass no elimina el directorio de almacenamiento anterior ni su contenido.
- Datos en tránsito local. AWS IoT Greengrass no cifra los datos de transmisión en tránsito local entre las fuentes de datos, AWS IoT Greengrass los componentes, el SDK de Stream Manager y el administrador de transmisiones.
- Datos en tránsito hacia. Nube de AWS Los flujos de datos exportados por el administrador de flujos para Nube de AWS utilizar el cifrado de cliente de AWS servicio estándar con Transport Layer Security (TLS).

Autenticación del cliente

Los clientes del administrador de flujos utilizan el SDK del administrador de flujos para comunicarse con el administrador de flujos. Cuando la autenticación de cliente está habilitada, solo los componentes de Greengrass pueden interactuar con los flujos en el administrador de flujos. Cuando la autenticación de cliente está deshabilitada, cualquier proceso que se ejecute en el dispositivo principal de Greengrass puede interactuar con los flujos en el administrador de flujos. Debe deshabilitar la autenticación solo si su caso de negocio lo requiere.

Utilice el parámetro STREAM_MANAGER_AUTHENTICATE_CLIENT para establecer el modo de autenticación del cliente. Puede configurar este parámetro cuando implementa el componente administrador de flujos en los dispositivos principales.

	Habilitado	Deshabilitado
Valor del parámetro	`true` (predeterminado y recomendado)	`false`
Clientes permitidos	Componentes de Greengrass en el dispositivo principal	Componentes de Greengrass en el dispositivo principal Otros procesos que se ejecutan en el dispositivo del núcleo de Greengrass

Véase también

- the section called “Configurar el administrador de secuencias”
- the section called “Se usa StreamManagerClient para trabajar con transmisiones”
- the section called “Exportación de configuraciones para destinos de nube compatibles”

Configurar el administrador de secuencias de AWS IoT Greengrass

En los dispositivos principales de Greengrass, el administrador de flujos puede almacenar, procesar y exportar datos enviados desde dispositivos IoT. El administrador de flujos proporciona parámetros que se usan para configurar los ajustes de tiempo de ejecución. Esta configuración se aplica a todos los flujos del dispositivo principal de Greengrass. Puede usar la consola o la API de AWS IoT Greengrass para configurar los ajustes del administrador del flujos cuando implemente el componente. Los cambios surten efecto después de que se completa la implementación.

Parámetros del administrador de secuencias

El administrador de flujos proporciona los siguientes parámetros que puede configurar al implementar el componente en sus dispositivos principales. Todos los parámetros son opcionales.

Directorio de almacenamiento

Nombre del parámetro: `STREAM_MANAGER_STORE_ROOT_DIR`

La ruta absoluta de la carpeta local usada para almacenar flujos. Este valor debe comenzar con una barra inclinada (por ejemplo, `/data`).

Debe especificar una carpeta existente y el usuario del sistema que ejecuta el componente de administrador de flujos debe tener permisos para leer y escribir en esta carpeta. Por ejemplo, puede ejecutar los siguientes comandos para crear y configurar una carpeta, `/var/greengrass/streams`, que especifique como carpeta raíz del administrador de flujos. Estos comandos permiten al usuario predeterminado del sistema, `ggc_user`, leer y escribir en esta carpeta.

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Para obtener información sobre cómo proteger los datos de secuencias, consulte the section called “Seguridad de los datos locales”.

Valor predeterminado: *`/greengrass/v2`*`/work/aws.greengrass.StreamManager`

Puerto del servidor

Nombre del parámetro: `STREAM_MANAGER_SERVER_PORT`

El número de puerto local utilizado para comunicarse con el administrador de secuencias. El valor predeterminado es `8088`.

Puede especificar `0` para usar un puerto disponible asignado al azar.

Autenticar cliente

Nombre del parámetro: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Indica si los clientes deben autenticarse para interactuar con el administrador de secuencias. El SDK del administrador de flujos controla toda la interacción entre los clientes y el administrador de flujos. Este parámetro determina qué clientes pueden llamar al SDK del administrador de flujos para trabajar con flujos. Para obtener más información, consulte the section called “Autenticación del cliente”.

Los valores válidos son `true` o `false`. El valor predeterminado es `true` (recomendado).

- `true`. Solo permite como clientes a los componentes de Greengrass. Los componentes usan protocolos de AWS IoT Greengrass Core internos para autenticarse con el SDK del administrador de flujos.

- `false`. Permite que cualquier proceso que se ejecute en el AWS IoT Greengrass Core sea un cliente. No establezca el valor en `false`, a menos que su caso de negocio lo requiera. Por ejemplo, use `false` solo si los procesos no componentes en el dispositivo principal deben comunicarse directamente con el administrador de flujos.

Ancho de banda máximo

Nombre del parámetro: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

El ancho de banda máximo promedio (en kilobits por segundo) que se puede utilizar para exportar datos. El valor predeterminado permite el uso ilimitado del ancho de banda disponible.

Tamaño del grupo de subprocesos

Nombre del parámetro: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

Cantidad máxima de subprocesos activos que se pueden utilizar para exportar datos. El valor predeterminado es `5`.

El tamaño óptimo depende del hardware, el volumen de secuencias y la cantidad planificada de secuencias de exportación. Si la velocidad de exportación es lenta, puede ajustar esta configuración para encontrar el tamaño óptimo para su hardware y su caso de negocio. La CPU y la memoria del hardware del dispositivo principal son factores limitantes. Para comenzar, puede intentar establecer este valor igual a la cantidad de núcleos de procesador en el dispositivo.

Tenga cuidado de no establecer un tamaño superior al que admite el hardware. Cada flujo utiliza recursos de hardware, por lo que debe intentar limitar la cantidad de flujos de exportación en dispositivos restringidos.

Argumentos de JVM

Nombre del parámetro: `JVM_ARGS`

Argumentos personalizados de la máquina virtual de Java para pasar al administrador de secuencias al inicio. Varios argumentos deben separarse por espacios.

Utilice este parámetro sólo cuando deba anular la configuración predeterminada utilizada por la JVM. Por ejemplo, puede que necesite aumentar el tamaño predeterminado del montón si planea exportar un gran número de secuencias.

Nivel de registro

Nombre del parámetro: `LOG_LEVEL`

El nivel de registro del componente. Elija uno de los siguientes niveles de registro, que se enumeran aquí en orden de niveles:

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Valor predeterminado: `INFO`

Tamaño mínimo para la carga de varias partes

Nombre del parámetro: `STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

El tamaño mínimo (en bytes) de una parte en una carga multiparte a Amazon S3. El administrador de flujos utiliza esta configuración y el tamaño del archivo de entrada para determinar cómo agrupar los datos en una solicitud PUT de varias partes. El valor predeterminado y mínimo es de `5242880` bytes (5 MB).

> **Note**
>
> El administrador de flujos usa la propiedad `sizeThresholdForMultipartUploadBytes` de la transmisión para determinar si se debe exportar a Amazon S3 como una carga única o multiparte. Los componentes de Greengrass definidos por el usuario establecen este umbral cuando crean un flujo que se exporta a Amazon S3. El umbral de tamaño predeterminado es 5 MB.

Véase también

- Administración de flujos de datos en los dispositivos principales de Greengrass
- Se usa StreamManagerClient para trabajar con transmisiones
- Exportación de configuraciones para Nube de AWS destinos compatibles

Creación de componentes personalizados que usen el administrador de flujos

Utilice el administrador de flujos en componentes personalizados de Greengrass para almacenar, procesar y exportar datos de dispositivos IoT. Utilice los procedimientos y ejemplos de esta sección para crear recetas de componentes, artefactos y aplicaciones que funcionen con el administrador de flujos. Para obtener más información sobre cómo desarrollar y probar componentes, consulte Creación de componentes de AWS IoT Greengrass.

Temas

- Definición de las recetas de componentes que utilizan el administrador de flujos
- Conexión al administrador de flujos en el código de la aplicación

Definición de las recetas de componentes que utilizan el administrador de flujos

Para usar el administrador de flujos en un componente personalizado, debe definir el componente `aws.greengrass.StreamManager` como una dependencia. También debe proporcionar el SDK del administrador de flujos. Complete las siguientes tareas para descargar y usar el SDK del administrador de flujos en el lenguaje que prefiera.

Uso del SDK del administrador de flujos para Java

El SDK del administrador de flujos para Java está disponible como un archivo JAR que puede usar para compilar su componente. A continuación, puede crear un JAR de aplicación que incluya el SDK del administrador de flujos, definir el JAR de aplicación como un artefacto de componente y ejecutar el JAR de aplicación en el ciclo de vida del componente.

Uso del SDK del administrador de flujos para Java

1. Descargue el archivo JAR del SDK del administrador de flujos para Java.
2. Realice una de las siguientes acciones para crear artefactos componentes a partir de su aplicación Java y el archivo JAR del SDK del administrador de flujos:
   - Cree su aplicación como un archivo JAR que incluya el JAR del SDK del administrador de flujos y ejecute este archivo JAR en la receta de su componente.

- Defina el JAR del SDK del administrador de flujos como un artefacto componente. Agregue ese artefacto a la ruta de clases cuando ejecute su aplicación en la receta de su componente.

La receta de su componente podría parecerse al siguiente ejemplo. Este componente ejecuta una versión modificada del ejemplo de StreamManagerS3.java, que `StreamManagerS3.jar` incluye el JAR del SDK de Stream Manager.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "Run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
```

```
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      Run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

Para obtener más información sobre cómo desarrollar y probar componentes, consulte Creación de componentes de AWS IoT Greengrass.

Uso del SDK del administrador de flujos para Python

El SDK del administrador de flujos para Python está disponible como código de origen que puede incluir en su componente. Cree un archivo ZIP del SDK del administrador de flujo, defina el archivo ZIP como un artefacto de un componente e instale los requisitos del SDK en el ciclo de vida del componente.

Uso del SDK del administrador de flujos para Python

1. Clona o descarga el repositorio aws-greengrass-stream-manager-sdk-python.

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-python.git
   ```

2. Cree un archivo ZIP que contenga la carpeta `stream_manager`, que incluye el código de origen del SDK del administrador de flujos para Python. Puede proporcionar este archivo ZIP como un artefacto componente que el software AWS IoT Greengrass Core descomprimirá al instalar el componente. Haga lo siguiente:

   a. Abra la carpeta que contiene el repositorio que clonó o descargó en el paso anterior.

      ```
      cd aws-greengrass-stream-manager-sdk-python
      ```

b. Comprima la carpeta `stream_manager` en un archivo ZIP llamado `stream_manager_sdk.zip`.

Linux or Unix

```
zip -rv stream_manager_sdk.zip stream_manager
```

Windows Command Prompt (CMD)

```
tar -acvf stream_manager_sdk.zip stream_manager
```

PowerShell

```
Compress-Archive stream_manager stream_manager_sdk.zip
```

c. Compruebe que el archivo `stream_manager_sdk.zip` contiene la carpeta `stream_manager` y su contenido. Ejecute el siguiente comando para hacer una lista del contenido del archivo ZIP.

Linux or Unix

```
unzip -l stream_manager_sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream_manager_sdk.zip
```

El resultado de debería parecerse al siguiente.

```
Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 20:45   stream_manager/
      913  02-24-2021 20:45   stream_manager/__init__.py
     9719  02-24-2021 20:45   stream_manager/utilinternal.py
     1412  02-24-2021 20:45   stream_manager/exceptions.py
     1004  02-24-2021 20:45   stream_manager/util.py
        0  02-24-2021 20:45   stream_manager/data/
   254463  02-24-2021 20:45   stream_manager/data/__init__.py
```

```
    26515  02-24-2021 20:45   stream_manager/streammanagerclient.py
---------                     -------
   294026                     8 files
```

3. Copie los artefactos del SDK del administrador de flujos a la carpeta de artefactos de su componente. Además del archivo ZIP del SDK del administrador de flujos, su componente utiliza el archivo `requirements.txt` del SDK para instalar las dependencias del SDK del administrador de flujos. *`~/greengrass-components`*Sustitúyalo por la ruta a la carpeta que utilizas para el desarrollo local.

   Linux or Unix

   ```
   cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
   com.example.StreamManagerS3Python/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 requirements.txt
   ```

   PowerShell

   ```
   cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0\
   ```

4. Cree su receta de componentes. En la receta, haga lo siguiente:

   a. Defina `stream_manager_sdk.zip` y `requirements.txt` como artefactos.

   b. Defina su aplicación de Python como un artefacto.

   c. En el ciclo de vida de la instalación, instale los requisitos del SDK del administrador de flujos desde `requirements.txt`.

   d. En el ciclo de vida de ejecución, agregue el SDK del administrador de flujos a `PYTHONPATH` y ejecute la aplicación Python.

   La receta de su componente podría parecerse al siguiente ejemplo. Este componente ejecuta el ejemplo de stream_manager_s3.py.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Python",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "Run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
```

```
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "Run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      Run: |
```

```
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      Run: |
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

Para obtener más información sobre cómo desarrollar y probar componentes, consulte Creación de componentes de AWS IoT Greengrass.

Usa el SDK de Stream Manager para JavaScript

El SDK de Stream Manager JavaScript está disponible como código fuente que puede incluir en su componente. Cree un archivo ZIP del SDK del administrador de flujo, defina el archivo ZIP como un artefacto de un componente e instale el SDK en el ciclo de vida del componente.

Para usar el SDK de Stream Manager para JavaScript

1. Clona o descarga el repositorio aws-greengrass-stream-manager-sdk-js.

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
```

2. Crea un archivo ZIP que contenga la `aws-greengrass-stream-manager-sdk` carpeta, que contiene el código fuente del SDK de Stream Manager para. JavaScript Puedes proporcionar este archivo ZIP como un artefacto componente que el software AWS IoT Greengrass principal descomprimirá al instalar el componente. Haga lo siguiente:

   a. Abra la carpeta que contiene el repositorio que clonó o descargó en el paso anterior.

   ```
   cd aws-greengrass-stream-manager-sdk-js
   ```

   b. Comprima la carpeta `aws-greengrass-stream-manager-sdk` en un archivo ZIP llamado `stream-manager-sdk.zip`.

   Linux or Unix

   ```
   zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
   ```

   Windows Command Prompt (CMD)

   ```
   tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
   ```

   PowerShell

   ```
   Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
   ```

   c. Compruebe que el archivo `stream-manager-sdk.zip` contiene la carpeta `aws-greengrass-stream-manager-sdk` y su contenido. Ejecute el siguiente comando para hacer una lista del contenido del archivo ZIP.

   Linux or Unix

   ```
   unzip -l stream-manager-sdk.zip
   ```

   Windows Command Prompt (CMD)

   ```
   tar -tf stream-manager-sdk.zip
   ```

El resultado de debería parecerse al siguiente.

```
Archive:  stream-manager-sdk.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/
      369  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/package.json
     1017  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/util.js
     8374  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/utilInternal.js
     1937  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/exceptions.js
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/
   353343  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/index.js
    22599  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/client.js
      216  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/index.js
---------                     -------
   387855                     9 files
```

3. Copie el artefacto del SDK del administrador de flujos a la carpeta de artefactos de su componente. *~/greengrass-components*Sustitúyalo por la ruta a la carpeta que utilizas para el desarrollo local.

   Linux or Unix

   ```
   cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
   com.example.StreamManagerS3JS/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
   ```

   PowerShell

   ```
   cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0\
   ```

4. Cree su receta de componentes. En la receta, haga lo siguiente:

   a. Defina `stream-manager-sdk.zip` como un artefacto.

   b. Defina su JavaScript aplicación como un artefacto.

c. En el ciclo de vida de la instalación, instale el SDK del administrador de flujos desde el artefacto `stream-manager-sdk.zip`. Este comando `npm install` crea una carpeta `node_modules` que contiene el SDK del administrador de flujos y sus dependencias.

d. En el ciclo de vida de ejecución, añada la `node_modules` carpeta a `NODE_PATH` la aplicación y JavaScript ejecútela.

La receta de su componente podría parecerse al siguiente ejemplo. Este componente ejecuta el ejemplo de StreamManagerS3.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3JS",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "Run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
```

```
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "Run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
```

```
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      Run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      Run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

Para obtener más información sobre cómo desarrollar y probar componentes, consulte Creación de componentes de AWS IoT Greengrass.

Conexión al administrador de flujos en el código de la aplicación

Para conectarse al administrador de flujos de su aplicación, cree una instancia `StreamManagerClient` desde el SDK del administrador de flujos. Este cliente se conecta al componente del administrador de flujos en su puerto predeterminado 8088 o en el puerto que especifique. Para obtener más información sobre cómo usar `StreamManagerClient` después de crear una instancia, consulte Se usa StreamManagerClient para trabajar con transmisiones.

Example Ejemplo: conectarse al administrador de flujos con el puerto predeterminado

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;

public class MyStreamManagerComponent {

    void connectToStreamManagerWithDefaultPort() {
        StreamManagerClient client = StreamManagerClientFactory.standard().build();

        // Use the client.
    }
}
```

Python

```
from stream_manager import (
    StreamManagerClient
)

def connect_to_stream_manager_with_default_port():
    client = StreamManagerClient()

    # Use the client.
```

JavaScript

```
const {
    StreamManagerClient
} = require('aws-greengrass-stream-manager-sdk');

function connectToStreamManagerWithDefaultPort() {
    const client = new StreamManagerClient();

    // Use the client.
}
```

Example Ejemplo: conectarse al administrador de flujos con un puerto no predeterminado

Si configura el administrador de flujos con un puerto distinto del predeterminado, debe utilizar la comunicación entre procesos para recuperar el puerto de la configuración del componente.

> **Note**
>
> El parámetro `port` de configuración contiene el valor que se especifica en `STREAM_MANAGER_SERVER_PORT` al implementar el administrador de flujos.

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
 IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
 GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
            greengrassCoreIPCClient.getConfiguration(request,
 Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

 .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
 StreamManagerClientFactory.standard().withClientConfig(config).build();

    // Use the client.
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
    StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

Se usa StreamManagerClient para trabajar con transmisiones

Los componentes de Greengrass definidos por el usuario que se ejecutan en el dispositivo principal de Greengrass pueden utilizar el objeto `StreamManagerClient` en el SDK del Administrador de flujos para crear flujos en el administrador de flujos y luego interactuar con ellos. Cuando un componente crea una transmisión, define los Nube de AWS destinos, la priorización y otras políticas de exportación y retención de datos para la transmisión. Para enviar datos al administrador de flujos, los componentes anexan los datos al flujo. Si se define un destino de exportación para el flujo, el administrador de flujos exporta la secuencia automáticamente.

> **Note**
>
> Normalmente, los clientes del administrador de flujos son componentes de Greengrass definidos por el usuario. Si su caso de negocio lo requiere, puede permitir que los procesos que no sean componentes que se ejecutan en el núcleo de Greengrass (por ejemplo, un contenedor de Docker) interactúen con el administrador de flujos. Para obtener más información, consulte the section called “Autenticación del cliente”.

Los fragmentos de este tema muestran cómo los clientes llaman a los métodos de `StreamManagerClient` para trabajar con secuencias. Para obtener detalles sobre la implementación de los métodos y sus argumentos, utilice los vínculos a la referencia del SDK de cada fragmento.

Si utiliza el administrador de flujos en una función de Lambda, la función de Lambda debe iniciar `StreamManagerClient` fuera del controlador de funciones. Si se crea una instancia en el controlador, la función crea un `client` y una conexión al administrador de secuencias cada vez que se invoca.

> **Note**
>
> Si crea una instancia `StreamManagerClient` en el controlador, debe llamar explícitamente al método `close()` cuando el `client` complete su trabajo. De lo contrario, el `client` mantiene la conexión abierta y otro subproceso en ejecución hasta que salga del script.

`StreamManagerClient` admite las siguientes operaciones:

- the section called “Creación de una secuencia de mensajes”
- the section called “Agregar un mensaje”
- the section called “Lectura de mensajes”
- the section called “Lista de secuencias”
- the section called “Descripción de una secuencia de mensajes”
- the section called “Actualización de un flujo de mensajes”
- the section called “Eliminación de una secuencia de mensajes”

Creación de una secuencia de mensajes

Para crear un flujo, un componente de Greengrass definido por el usuario llama al método de creación y pasa un objeto `MessageStreamDefinition`. Este objeto especifica el nombre exclusivo del flujo y define cómo el administrador del flujo debe gestionar los nuevos datos cuando se alcanza el tamaño máximo de flujo. Puede utilizar `MessageStreamDefinition` y sus tipos de datos (como `ExportDefinition`, `StrategyOnFull` y `Persistence`) para definir otras propiedades de secuencias. Entre ellos se incluyen:

- El destino AWS IoT Analytics, Kinesis Data Streams AWS IoT SiteWise y Amazon S3 para las exportaciones automáticas. Para obtener más información, consulte the section called “Exportación de configuraciones para destinos de nube compatibles”.
- Prioridad de exportación. El administrador de secuencias exporta secuencias de mayor prioridad antes que secuencias de menor prioridad.
- Tamaño e intervalo de lote máximos para AWS IoT Analytics Kinesis Data Streams AWS IoT SiteWise y destinos. El administrador de secuencias exporta mensajes cuando se cumple cualquiera de las condiciones.
- Time-to-live (TTL). La cantidad de tiempo para garantizar que los datos de secuencia están disponibles para su procesamiento. Debe asegurarse de que los datos se pueden consumir dentro de este período de tiempo. Esta no es una política de eliminación. Es posible que los datos no se eliminen inmediatamente después del período TTL.
- Persistencia de secuencia. Elija guardar las secuencias en el sistema de archivos para conservar los datos en los reinicios del núcleo o guardar las secuencias en la memoria.
- Inicio del número de secuencia Especifique el número de secuencia del mensaje que se utilizará como mensaje de inicio en la exportación.

Para obtener más información acerca de `MessageStreamDefinition`, consulte la referencia del SDK para el lenguaje de destino:

- MessageStreamDefinitionen el SDK de Java
- MessageStreamDefinitionen el SDK de Node.js
- MessageStreamDefinitionen el SDK de Python

> **Note**
>
> `StreamManagerClient` también proporciona un destino que puede utilizar para exportar secuencias a un servidor HTTP. Este destino está pensado solo con fines de prueba. No es estable y no se admite para su uso en entornos de producción.

Una vez creado un flujo, los componentes de Greengrass pueden anexar mensajes al flujo con el fin de enviar datos para su exportación y leer los mensajes del flujo para su procesamiento local. El número de secuencias que cree depende de sus capacidades de hardware y de su caso de negocio. Una estrategia consiste en AWS IoT Analytics crear una transmisión para cada canal de destino de la transmisión de datos de Kinesis, aunque puede definir varios destinos para una transmisión. Una secuencia tiene una vida útil duradera.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

El siguiente fragmento crea una secuencia denominada `StreamName`. Define las propiedades de las secuencias en `MessageStreamDefinition` y los tipos de datos subordinados.

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",    # Required.
        max_size=268435456,    # Default is 256 MB.
        stream_segment_size=16777216,    # Default is 16 MB.
        time_to_live_millis=None,    # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,    # Required.
        persistence=Persistence.File,    # Default is File.
        flush_on_write=False,    # Default is false.
        export_definition=ExportDefinition(    # Optional. Choose where/how the
 stream is exported to the Nube de AWS.
```

```
            kinesis=None,
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: create_message_stream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)    // Default is 256 MB.
                    .withStreamSegmentSize(16777216L)    // Default is 16 MB.
                    .withTimeToLiveMillis(null)    // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)    //
 Required.
                    .withPersistence(Persistence.File)    // Default is File.
                    .withFlushOnWrite(false)    // Default is false.
                    .withExportDefinition(    // Optional. Choose where/how the
 stream is exported to the Nube de AWS.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del createMessageStreamSDK de Java: | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
  try {
    await client.createMessageStream(
      new MessageStreamDefinition()
        .withName("StreamName") // Required.
        .withMaxSize(268435456)  // Default is 256 MB.
        .withStreamSegmentSize(16777216)  // Default is 16 MB.
        .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
        .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  // Required.
        .withPersistence(Persistence.File)  // Default is File.
        .withFlushOnWrite(false)  // Default is false.
        .withExportDefinition(  // Optional. Choose where/how the stream is exported
 to the Nube de AWS.
          new ExportDefinition()
            .withKinesis(null)
            .withIotAnalytics(null)
            .withIotSiteWise(null)
            .withS3(null)
        )
    );
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: createMessageStream| MessageStreamDefinition

Para obtener más información acerca de la configuración de destinos de exportación, consulte the section called “Exportación de configuraciones para destinos de nube compatibles”.

Agregar un mensaje

Para enviar datos al administrador de flujos para su exportación, los componentes de Greengrass anexan los datos al flujo de destino. El destino de exportación determina el tipo de datos que se van a transferir a este método.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

AWS IoT Analytics o destinos de exportación de Kinesis Data Streams

El siguiente fragmento agrega un mensaje a la secuencia denominada `StreamName`. Para los AWS IoT Analytics destinos de Kinesis Data Streams, sus componentes de Greengrass añaden una masa de datos.

Este fragmento tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
 data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: append_message

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
```

```
    // Properly handle exception.
}
```

Referencia de SDK de Java: appendMessage

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const sequenceNumber = await client.appendMessage("StreamName",
 Buffer.from("Arbitrary byte array"));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: appendMessage

AWS IoT SiteWise destinos de exportación

El siguiente fragmento agrega un mensaje a la secuencia denominada `StreamName`. Para AWS IoT SiteWise los destinos, sus componentes de Greengrass añaden un objeto serializado. `PutAssetPropertyValueEntry` Para obtener más información, consulte the section called "Exportación a AWS IoT SiteWise".

> **Note**
>
> Al enviar datos a AWS IoT SiteWise, estos deben cumplir los requisitos de la acción. `BatchPutAssetPropertyValue` Para obtener más información, consulta BatchPutAssetPropertyValue en la AWS IoT SiteWise Referencia de la API de .

Este fragmento tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
 not earlier
    # than 10 minutes in the past. Add some randomness to time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: [append_message | PutAssetPropertyValueEntry]

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
 in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;
```

```
    // IoTSiteWise requires unique timestamps in all messages and also needs
 timestamps not earlier
    // than 10 minutes in the past. Add some randomness to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: AppendMessage | PutAssetPropertyValueEntry

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
 defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
 Math.floor(Math.random() - maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
```

```
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);

        const putAssetPropertyValueEntry =    new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: AppendMessage | PutAssetPropertyValueEntry

Destinos de exportación de Amazon S3

El siguiente fragmento añade una tarea de exportación a la secuencia denominada `StreamName`. Para los destinos de Amazon S3, los componentes de Greengrass anexan un objeto `S3ExportTaskDefinition` serializado que contiene información sobre el archivo de entrada de origen y el objeto de Amazon S3 de destino. Si el objeto especificado no existe, el administrador de flujos lo crea por usted. Para obtener más información, consulte the section called “Exportación a Amazon S3”.

Este fragmento tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
```

```
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
 bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: append_message | S3 ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: AppendMessage | S3 ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
```

```
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: AppendMessage | S3 ExportTaskDefinition

Lectura de mensajes

Leer mensajes de un flujo.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

El siguiente fragmento lee los mensajes de la secuencia denominada `StreamName`. El método read toma un objeto `ReadMessagesOptions` opcional que especifica el número de secuencia para comenzar a leer, los números mínimo y máximo para leer y un tiempo de espera para leer mensajes.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
 the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
 0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is raised. By default, this is 1.
```

```
            max_message_count=100,    # Accept up to 100 messages. By default this
 is 1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_messsage_count to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: read_messages | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<Message> messages = client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: readMessages | ReadMessagesOptions

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messages = await client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                // Try to read from sequence number 100 or greater. By default this
 is 0.
                .withDesiredStartSequenceNumber(100)
                // Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is thrown. By default, this is 1.
                .withMinMessageCount(10)
                // Accept up to 100 messages. By default this is 1.
                .withMaxMessageCount(100)
                // Try to wait at most 5 seconds for the minMessageCount to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                .withReadTimeoutMillis(5 * 1000)
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: readMessages | ReadMessagesOptions

Lista de secuencias

Obtenga la lista de flujos en el administrador de flujos.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

El siguiente fragmento de código obtiene una lista de las secuencias (por nombre) del administrador de secuencias.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: list_streams

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia de SDK de Java: listStreams

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const streams = await client.listStreams();
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
```

```
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del Node.js SDK: listStreams

Descripción de una secuencia de mensajes

Obtenga metadatos sobre un flujo, incluida la definición, el tamaño y el estado de exportación del flujo.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

El siguiente fragmento de código obtiene metadatos de una secuencia llamada `StreamName`, como su definición, su tamaño y los estados del exportador.

Python

```
client = StreamManagerClient()

try:
    stream_description = client.describe_message_stream(stream_name="StreamName")
    if stream_description.export_statuses[0].error_message:
        # The last export of export destination 0 failed with some error
        # Here is the last sequence number that was successfully exported
        stream_description.export_statuses[0].last_exported_sequence_number

    if (stream_description.storage_status.newest_sequence_number >
            stream_description.export_statuses[0].last_exported_sequence_number):
        pass
        # The end of the stream is ahead of the last exported sequence number
except StreamManagerException:
    pass
    # Properly handle errors.
```

```
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: describe_message_stream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
            description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: describeMessageStream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
```

```
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: describeMessageStream

Actualización de un flujo de mensajes

Actualiza las propiedades de un flujo existente. Es posible que desee actualizar un flujo si sus requisitos cambian después de crearlo. Por ejemplo:

- Agrega una nueva configuración de exportación para un Nube de AWS destino.
- Aumente el tamaño máximo de un flujo para cambiar la forma en que se exportan o conservan los datos. Por ejemplo, el tamaño del flujo, en combinación con su estrategia en la configuración completa, puede provocar que los datos se eliminen o rechacen antes de que el administrador de flujos pueda procesarlos.
- Pausa y reanuda las exportaciones; por ejemplo, si las tareas de exportación son prolongadas y quiere racionar los datos de carga.

Los componentes de Greengrass siguen este proceso general para actualizar un flujo:

1. Consiga la descripción del flujo.
2. Actualice las propiedades de destino de los objetos correspondientes `MessageStreamDefinition` y subordinados.
3. Transfiera la actualización `MessageStreamDefinition`. Asegúrese de incluir las definiciones de objetos completas para el flujo actualizado. Las propiedades no definidas revierten a los valores predeterminados.

   Puede especificar el número de secuencia del mensaje que se utilizará como mensaje de inicio en la exportación.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

El siguiente fragmento de código actualiza la secuencia llamada `StreamName`. Actualiza varias propiedades de un flujo que se exporta a Kinesis Data Streams.

Python

```
client = StreamManagerClient()

try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=
        [KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: updateMessageStream| MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
```

```
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
           // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)    // Update TTL to 1 hour.
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the Nube de
 AWS.
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: update_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)    // Default is 256 MB. Updating Max Size
 to 512 MB.
                .withStreamSegmentSize(33554432)    // Default is 16 MB. Updating
 Segment Size to 32 MB.
```

```
                .withTimeToLiveMillis(3600000)    // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)    // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)    // Default is File. Update
 the persistence to Memory
                .withFlushOnWrite(true)    // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the Nube
 de AWS.
                    messageStreamInfo.definition.exportDefinition
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: | updateMessageStreamMessageStreamDefinition

Restricciones para actualizar los flujos

Se aplican las siguientes restricciones al actualizar flujos. A menos que se indique en la siguiente lista, las actualizaciones se aplican de inmediato.

- No puede actualizar la persistencia de un flujo. Para cambiar este comportamiento, elimine el flujo y crea un flujo que defina la nueva política de persistencia.
- Puede actualizar el tamaño máximo de una transmisión solo en las siguientes condiciones:
  - El tamaño máximo debe ser superior o igual al tamaño actual del flujo. Para encontrar esta información, describa la secuencia y, a continuación, compruebe el estado de almacenamiento del objeto `MessageStreamInfo` devuelto.
  - El tamaño máximo debe ser superior o igual al tamaño del segmento del flujo.

- Puede actualizar el tamaño del segmento de la transmisión a un valor inferior al tamaño máximo del flujo. La configuración actualizada se aplica a los segmentos nuevos.
- Las actualizaciones de la propiedad tiempo de vida (TTL) se aplican a las nuevas operaciones de anexión. Si reduce este valor, es posible que el administrador de flujos también elimine los segmentos existentes que superen el TTL.
- Las actualizaciones de la estrategia en toda la propiedad se aplican a las nuevas operaciones de anexión. Si establece la estrategia para sobrescribir los flujos de datos más antiguos, es posible que el administrador de flujos también sobrescriba los segmentos existentes en función del nuevo ajuste.
- Las actualizaciones de la propiedad de vaciar al escribir se aplican a los mensajes nuevos.
- Las actualizaciones de las configuraciones de exportación se aplican a las nuevas exportaciones. La solicitud de actualización debe incluir todas las configuraciones de exportación que desee admitir. De lo contrario, el administrador de flujos las elimina.
  - Al actualizar una configuración de exportación, especifique el identificador de la configuración de exportación de destino.
  - Para añadir una configuración de exportación, especifique un identificador único para la nueva configuración de exportación.
  - Para eliminar una configuración de exportación, omita la configuración de exportación.
- Para actualizar el número de secuencia inicial de una configuración de exportación en un flujo, debe especificar un valor inferior al último número de secuencia. Para encontrar esta información, describa la secuencia y, a continuación, compruebe el estado de almacenamiento del objeto `MessageStreamInfo` devuelto.

Eliminación de una secuencia de mensajes

Elimina un flujo. Cuando elimina una secuencia, todos los datos almacenados para la secuencia se eliminan del disco.

Requisitos

Esta operación tiene los siguientes requisitos:

- Versión mínima del SDK del Administrados de flujos: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Ejemplos

El siguiente fragmento de código elimina la secuencia llamada `StreamName`.

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: deleteMessageStream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: delete_message_stream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.deleteMessageStream("StreamName");
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
```

```
});
```

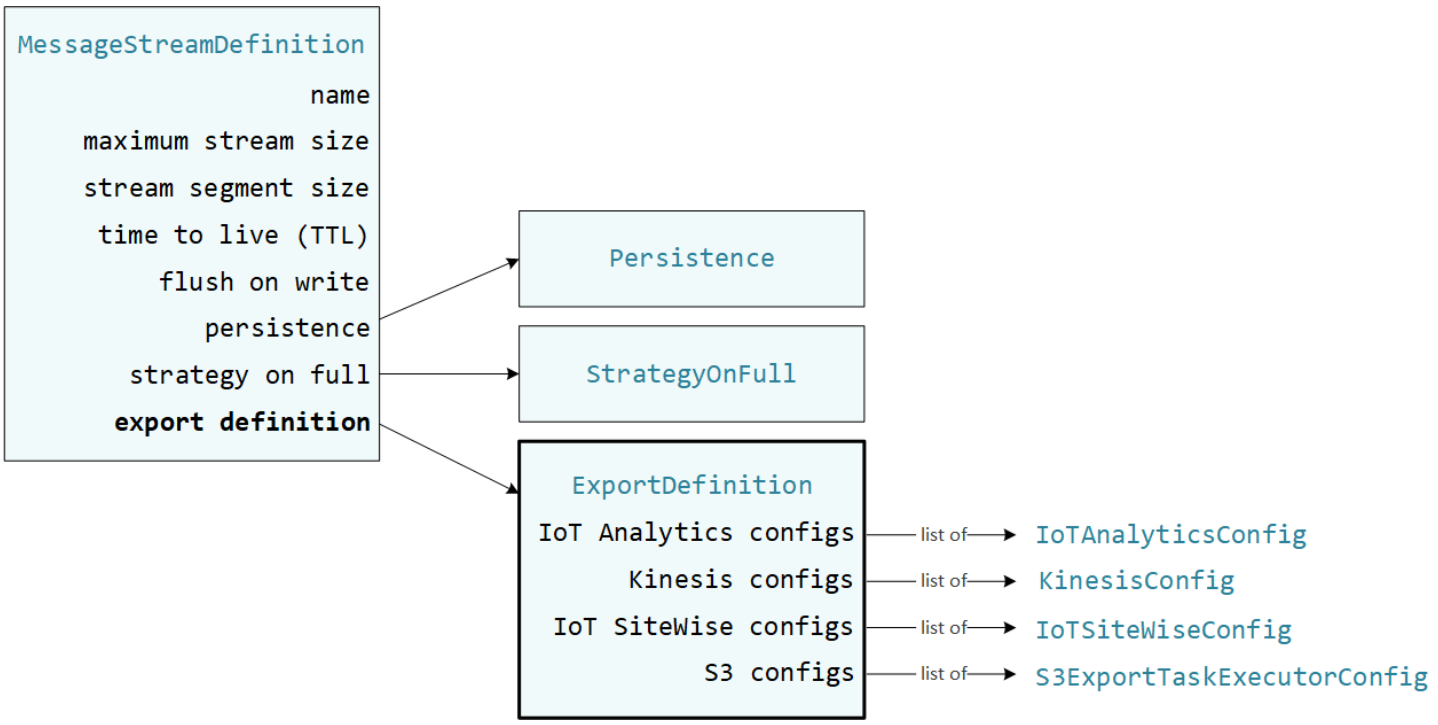
Referencia del SDK de Node.js: deleteMessageStream

Véase también

- Administración de flujos de datos en los dispositivos principales de Greengrass
- Configurar el administrador de secuencias de AWS IoT Greengrass
- Exportación de configuraciones para Nube de AWS destinos compatibles
- `StreamManagerClient` en la referencia del SDK del Administrador de flujos:
  - Python
  - Java
  - Node.js

Exportación de configuraciones para Nube de AWS destinos compatibles

Los componentes de Greengrass definidos por el usuario utilizan `StreamManagerClient` en el SDK del administrador de flujos para interactuar con el administrador de flujos. Cuando un componente crea un flujo o actualiza un flujo, pasa un objeto `MessageStreamDefinition` que representa las propiedades del flujo, incluida la definición de exportación. El objeto `ExportDefinition` contiene las configuraciones de exportación definidas para el flujo. El administrador de flujos utiliza estas configuraciones de exportación para determinar dónde y cómo exportar el flujo.



Puede definir cero o más configuraciones de exportación en un flujo, incluidas varias configuraciones de exportación para un único tipo de destino. Por ejemplo, puede exportar un flujo a dos canales AWS IoT Analytics y a un flujo de datos de Kinesis.

En caso de intentos de exportación fallidos, Stream Manager vuelve a intentar exportar los datos continuamente a Nube de AWS intervalos de hasta cinco minutos. La cantidad de reintentos no tiene un límite máximo.

> **Note**
>
> `StreamManagerClient` también proporciona un destino que puede utilizar para exportar secuencias a un servidor HTTP. Este destino está pensado solo con fines de prueba. No es estable y no se admite para su uso en entornos de producción.

Destinos compatibles Nube de AWS

- AWS IoT Analytics canales
- Amazon Kinesis Data Streams
- AWS IoT SiteWise propiedades de los activos
- Objetos de Amazon S3

Usted es responsable del mantenimiento de estos Nube de AWS recursos.

AWS IoT Analytics canales

El administrador de transmisiones admite exportaciones automáticas a AWS IoT Analytics. AWS IoT Analytics le permite realizar análisis avanzados de sus datos para ayudarle a tomar decisiones empresariales y mejorar los modelos de aprendizaje automático. Para obtener más información, consulte ¿Qué es AWS IoT Analytics? en la Guía AWS IoT Analytics del usuario.

En el SDK del administrador de flujos, los componentes de Greengrass utilizan `IoTAnalyticsConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- Io TAnalytics Config en el SDK de Python
- Io TAnalytics Config en el SDK de Java
- Io TAnalytics Config en el SDK de Node.js

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Los canales de destino AWS IoT Analytics deben estar en el mismo Cuenta de AWS y Región de AWS igual que el dispositivo principal de Greengrass.
- El Autorizar a los dispositivos principales a interactuar con AWS los servicios debe permitir el permiso `iotanalytics:BatchPutMessage` para segmentar los canales. Por ejemplo:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_1_name",
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
      ]
```

```
    }
  ]
}
```

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Exportando a AWS IoT Analytics

Para crear una transmisión a la que se exporte AWS IoT Analytics, sus componentes de Greengrass crean una transmisión con una definición de exportación que incluye uno o más `IoTAnalyticsConfig` objetos. Este objeto define los ajustes de exportación, como el canal de destino, el tamaño del lote, el intervalo del lote y la prioridad.

Cuando sus componentes de Greengrass reciben datos de los dispositivos, anexan mensajes que contienen una masa de datos al flujo de destino.

A continuación, el administrador de flujos exporta los datos en función de los ajustes del lote y la prioridad definidos en las configuraciones de exportación del flujo.

Amazon Kinesis Data Streams

El administrador de flujos permite exportar automáticamente a Amazon Kinesis Data Streams. Kinesis Data Streams se utiliza normalmente para agregar grandes volúmenes de datos y cargarlos en un almacén MapReduce de datos o un clúster. Para obtener más información, consulte Qué son los Amazon Kinesis Data Streams en la Guía para desarrolladores de Amazon Kinesis.

En el SDK del administrador de flujos, los componentes de Greengrass utilizan `KinesisConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- KinesisConfigen el SDK de Python
- KinesisConfigen el SDK de Java
- KinesisConfigen el SDK de Node.js

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Las transmisiones de destino de Kinesis Data Streams deben estar en el Cuenta de AWS mismo dispositivo principal de Greengrass Región de AWS y en el mismo.
- (Recomendado) La versión 2.2.1 del administrador de flujos mejora el rendimiento de la exportación de flujos a los destinos del flujo de datos de Kinesis. Para usar las mejoras de esta última versión, actualice su componente de administrador de flujos a la versión 2.2.1 y utilice la política `kinesis:ListShards` en el rol de intercambio de tokens de Greengrass.
- El Autorizar a los dispositivos principales a interactuar con AWS los servicios debe conceder el permiso `kinesis:PutRecords` para destinar flujos de datos. Por ejemplo:

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "kinesis:PutRecords"
        ],
        "Resource": [
          "arn:aws:kinesis:us-east-1:123456789012:stream/stream_1_name",
          "arn:aws:kinesis:us-east-1:123456789012:stream/stream_2_name"
        ]
      }
    ]
  }
  ```

  Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Exportación a Kinesis Data Streams

Para crear un flujo que se exporte a Kinesis Data Streams, sus componentes de Greengrass crean un flujo con una definición de exportación que incluye uno o más objetos `KinesisConfig`. Este objeto define los ajustes de exportación, como el flujo de datos de destino, el tamaño del lote, el intervalo del lote y la prioridad.

Cuando sus componentes de Greengrass reciben datos de los dispositivos, anexan mensajes que contienen una masa de datos al flujo de destino. A continuación, el administrador de flujos

exporta los datos en función de los ajustes del lote y la prioridad definidos en las configuraciones de exportación del flujo.

El administrador de flujos genera un UUID aleatorio único como clave de partición para cada registro cargado en Amazon Kinesis.

AWS IoT SiteWise propiedades de los activos

Stream Manager admite la exportación automática a AWS IoT SiteWise. AWS IoT SiteWise le permite recopilar, organizar y analizar datos de equipos industriales a escala. Para obtener más información, consulte ¿Qué es AWS IoT SiteWise? en la Guía AWS IoT SiteWise del usuario.

En el SDK del administrador de flujos, los componentes de Greengrass utilizan `IoTSiteWiseConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- Io TSite WiseConfig en el SDK de Python
- Io TSite WiseConfig en el SDK de Java
- Io TSite WiseConfig en el SDK de Node.js

> **Note**
>
> AWS también proporciona AWS IoT SiteWise componentes, que ofrecen una solución prediseñada que puede utilizar para transmitir datos desde fuentes OPC-UA. Para obtener más información, consulte Colector IoT SiteWise OPC UA.

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Las propiedades de los activos de destino AWS IoT SiteWise deben estar en el mismo Cuenta de AWS y Región de AWS igual que el dispositivo principal de Greengrass.

  > **Note**
  >
  > Para ver la lista de Región de AWS dispositivos AWS IoT SiteWise compatibles, consulte los AWS IoT SiteWise puntos finales y las cuotas en la Referencia AWS general.

- Autorizar a los dispositivos principales a interactuar con AWS los serviciosEl `iotsitewise:BatchPutAssetPropertyValue` debe permitir que el permiso se dirija a las propiedades de los activos. El siguiente ejemplo de política utiliza la clave `iotsitewise:assetHierarchyPath` de condición para conceder acceso a un activo raíz de destino y a sus elementos secundarios. Puede eliminarlos `Condition` de la política para permitir el acceso a todos sus AWS IoT SiteWise activos o especificar ARNs activos individuales.

  JSON

  ```
  {
    "Version":"2012-10-17",
    "Statement": [
      {
         "Effect": "Allow",
         "Action": "iotsitewise:BatchPutAssetPropertyValue",
         "Resource": "*",
         "Condition": {
           "StringLike": {
             "iotsitewise:assetHierarchyPath": [
               "/root node asset ID",
               "/root node asset ID/*"
             ]
           }
         }
      }
    ]
  }
  ```

  Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

  Para obtener información de seguridad importante, consulte la BatchPutAssetPropertyValue autorización en la Guía del AWS IoT SiteWise usuario.

Exportación a AWS IoT SiteWise

Para crear una transmisión a la que se exporte AWS IoT SiteWise, sus componentes de Greengrass crean una transmisión con una definición de exportación que incluye uno o más

`IoTSiteWiseConfig` objetos. Este objeto define los ajustes de exportación, como el tamaño del lote, el intervalo del lote y la prioridad.

Cuando sus componentes de Greengrass reciben datos de propiedad de activos, anexan mensajes que contienen datos al flujo de destino. Los mensajes son objetos `PutAssetPropertyValueEntry` serializados en JSON que contienen los valores de propiedad de una o más propiedades de los activos. Para obtener más información, consulte Añadir un mensaje para los AWS IoT SiteWise destinos de exportación.

> **Note**
>
> Al enviar datos a AWS IoT SiteWise, estos deben cumplir los requisitos de la `BatchPutAssetPropertyValue` acción. Para obtener más información, consulta BatchPutAssetPropertyValue en la AWS IoT SiteWise Referencia de la API de .

A continuación, el administrador de flujos exporta los datos en función de los ajustes del lote y la prioridad definidos en las configuraciones de exportación del flujo.

Puede ajustar la configuración del administrador de flujos y la lógica del componente de Greengrass para diseñar su estrategia de exportación. Por ejemplo:

- Para realizar exportaciones prácticamente en tiempo real, establezca ajustes del tamaño de lote e intervalos bajos y añada los datos al flujo cuando lo reciba.
- Para optimizar el procesamiento por lotes, mitigar las restricciones de ancho de banda o minimizar los costos, sus componentes de Greengrass pueden agrupar timestamp-quality-value los puntos de datos (TQV) recibidos para una sola propiedad de activo antes de agregar los datos a la transmisión. Una estrategia consiste en agrupar las entradas de hasta 10 combinaciones diferentes de propiedades y activos (o alias de propiedades) en un mensaje, en lugar de enviar más de una entrada para la misma propiedad. Esto ayuda al Administrador de flujos a mantenerse dentro de las cuotas de AWS IoT SiteWise.

Objetos de Amazon S3

El administrador de flujos permite exportar automáticamente a Amazon S3. Puede utilizar Amazon S3 para almacenar y recuperar grandes cantidades de datos. Para obtener más información, consulte ¿Qué es Amazon S3? en la Guía para desarrolladores de Amazon Simple Storage Service.

En el SDK del administrador de flujos, los componentes de Greengrass utilizan `S3ExportTaskExecutorConfig` para definir la configuración de exportación para este tipo de destino. Para obtener más información, consulte la referencia del SDK para el lenguaje de destino.

- S3 ExportTaskExecutorConfig en el SDK de Python
- S3 ExportTaskExecutorConfig en el SDK de Java
- S3 ExportTaskExecutorConfig en el SDK de Node.js

Requisitos

Este destino de exportación tiene los siguientes requisitos:

- Los buckets Amazon S3 de destino deben estar en el mismo lugar Cuenta de AWS que el dispositivo principal de Greengrass.
- Si una función de Lambda que se ejecuta en el modo contenedor de Greengrass escribe archivos de entrada en el directorio de archivos de entrada, usted debe crear el directorio como un volumen en el contenedor con permisos de escritura. Esto garantiza que los archivos se escriban en el sistema de archivos raíz y que sean visibles en el componente administrador de flujos, que se ejecuta fuera del contenedor.
- Si un componente contenedor de Docker escribe archivos de entrada en un directorio de archivos de entrada, usted debe montar el directorio como un volumen en el contenedor con permisos de escritura. Esto garantiza que los archivos se escriban en el sistema de archivos raíz y que sean visibles en el componente administrador de flujos, que se ejecuta fuera del contenedor.
- El Autorizar a los dispositivos principales a interactuar con AWS los servicios debe permitir los siguientes permisos para los buckets de destino. Por ejemplo:

  JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
```

```
        "arn:aws:s3:::bucket-1-name/*",
        "arn:aws:s3:::bucket-2-name/*"
      ]
    }
  ]
}
```

Puede conceder acceso granular o condicional a recursos (por ejemplo, utilizando un esquema de nomenclatura con comodín *) Para obtener más información, consulte Adición y eliminación de políticas de IAM en la Guía del usuario de IAM.

Exportación a Amazon S3

Para crear un flujo que se exporte a Amazon S3, sus componentes de Greengrass utilizan el objeto `S3ExportTaskExecutorConfig` para configurar la política de exportación. La política define los ajustes de exportación, como el umbral de carga multiparte y la prioridad. Para las exportaciones de Amazon S3, el administrador de flujos carga los datos que lee de los archivos locales en el dispositivo principal. Para iniciar una carga, sus componentes de Greengrass anexan una tarea de exportación al flujo de destino. La tarea de exportación contiene información sobre el archivo de entrada y el objeto Amazon S3 de destino. El administrador de flujos ejecuta las tareas en la secuencia en que se anexan al flujo.

> **Note**
>
> El depósito de destino ya debe existir en su. Cuenta de AWS Si no existe un objeto para la clave especificada, el administrador de flujos crea el objeto automáticamente.

El administrador de flujos utiliza la propiedad de umbral de carga multiparte, el ajuste del tamaño mínimo de las piezas y el tamaño del archivo de entrada para determinar cómo cargar los datos. El umbral de carga multiparte debe ser igual o mayor que el tamaño mínimo de la pieza. Si desea cargar datos en paralelo, puede crear varios flujos.

Las claves que especifican los objetos de Amazon S3 de destino pueden incluir DateTimeFormatter cadenas Java válidas en `!{timestamp:value}` los marcadores de posición. Puede utilizar estos marcadores de fecha y hora para particionar los datos en Amazon S3 en función de la hora en la que se cargaron los datos del archivo de entrada. Por ejemplo, el siguiente nombre de clave se resuelve en un valor como `my-key/2020/12/31/data.txt`.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Si desea supervisar el estado de exportación de un flujo, cree primero un flujo de estado y, a continuación, configure el flujo de exportación para utilizarlo. Para obtener más información, consulte the section called “Supervise las tareas de exportación”.

Administración de datos de entrada

Puede crear un código que las aplicaciones de IoT usen para administrar el ciclo de vida de los datos de entrada. El siguiente ejemplo de flujo de trabajo muestra cómo se pueden utilizar los componentes de Greengrass para administrar estos datos.

1. Un proceso local recibe datos de dispositivos o periféricos y, a continuación, los escribe en los archivos de un directorio del dispositivo principal. Estos son los archivos de entrada del administrador de flujos.
2. Un componente de Greengrass escanea el directorio y anexa una tarea de exportación al flujo de destino cuando se crea un archivo nuevo. La tarea es un objeto `S3ExportTaskDefinition` serializado en JSON que especifica la URL del archivo de entrada, el bucket y la clave de Amazon S3 de destino y los metadatos de usuario opcionales.
3. El administrador de flujos lee el archivo de entrada y exporta los datos a Amazon S3 en el orden de las tareas anexas. El bucket de destino ya debe existir en su. Cuenta de AWS Si no existe un objeto para la clave especificada, el administrador de flujos crea el objeto automáticamente.
4. El componente de Greengrass lee los mensajes de un flujo de estado para supervisar el estado de la exportación. Una vez finalizadas las tareas de exportación, el componente de Greengrass puede eliminar los archivos de entrada correspondientes. Para obtener más información, consulte the section called “Supervise las tareas de exportación”.

Supervise las tareas de exportación

Puede crear un código que las aplicaciones de IoT utilizan para monitorear el estado de sus exportaciones de Amazon S3. Sus componentes de Greengrass deben crear un flujo de estado y, a continuación, configurar el flujo de exportación para escribir actualizaciones de estado en el flujo de estado. Una sola transmisión de estado puede recibir actualizaciones de estado de varias transmisiones que se exportan a Amazon S3.

En primer lugar, cree un flujo para utilizarlo como flujo de estado. Puede configurar las políticas de tamaño y retención del flujo para controlar la vida útil de los mensajes de estado. Por ejemplo:

- Configure `Persistence` en `Memory` si no desea guardar los mensajes de estado.
- Configure `StrategyOnFull` en `OverwriteOldestData` para que no se pierdan los nuevos mensajes de estado.

A continuación, cree o actualice el flujo de exportación para usar el flujo de estado.
En concreto, defina la propiedad de configuración de estado de la configuración de `S3ExportTaskExecutorConfig` exportación del flujo. Esto le indica al administrador del flujo que escriba mensajes de estado sobre las tareas de exportación en el flujo de estado. En el objeto `StatusConfig`, especifique el nombre del flujo de estado y el nivel de detalle. Los siguientes valores admitidos van desde el menos detallado (`ERROR`) al más detallado (). `TRACE` El valor predeterminado es `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`
- `TRACE`

El siguiente ejemplo de flujo de trabajo muestra cómo los componentes de Greengrass pueden utilizar un flujo de estado para supervisar el estado de exportación.

1. Como se describió en el flujo de trabajo anterior, un componente de Greengrass anexa una tarea de exportación a un flujo que está configurado para escribir mensajes de estado sobre las tareas de exportación en un flujo de estado. La operación de incorporación devuelve un número de secuencia que representa el ID de la tarea.
2. Un componente de Greengrass lee los mensajes secuencialmente del flujo de estado y, a continuación, filtra los mensajes en función del nombre del flujo y el identificador de la tarea o en función de una propiedad de la tarea de exportación del contexto del mensaje. Por ejemplo, el componente de Greengrass puede filtrar por la URL del archivo de entrada de la tarea de exportación, que está representada por el objeto `S3ExportTaskDefinition` en el contexto del mensaje.

Los siguientes códigos de estado indican que una tarea de exportación ha alcanzado un estado completo:

- `Success`. Se ha completado correctamente la carga.
- `Failure`. El administrador de flujos detectó un error; por ejemplo, el bucket especificado no existe. Tras resolver el problema, puede volver a añadir la tarea de exportación al flujo.
- `Canceled`. La tarea se detuvo porque se eliminó la definición de transmisión o exportación o porque el período time-to-live (TTL) de la tarea expiró.

> **Note**
>
> La tarea también puede tener un estado de `InProgress` o `Warning`. El administrador de flujos emite advertencias cuando un evento devuelve un error que no afecta a la ejecución de la tarea. Por ejemplo, si no se borra una carga parcial cancelada, aparecerá una advertencia.

3. Una vez finalizadas las tareas de exportación, el componente de Greengrass puede eliminar los archivos de entrada correspondientes.

En el siguiente ejemplo, se muestra cómo un componente de Greengrass puede leer y procesar los mensajes de estado.

Python

```
import time
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
```

```
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referencia del SDK de Python: read_messages | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

try (final StreamManagerClient client =
StreamManagerClientFactory.standard().build()) {
   try {
       boolean isS3UploadComplete = false;
       while (!isS3UploadComplete) {
           try {
               // Read the statuses from the export status stream
               List<Message> messages = client.readMessages("StatusStreamName",
                   new
ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
               for (Message message : messages) {
                   // Deserialize the status message first.
                   StatusMessage statusMessage =
ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
StatusMessage.class);
                   // Check the status of the status message. If the status is
"Success", the file was successfully uploaded to S3.
                   // If the status was either "Failure" or "Canceled", the server
was unable to upload the file to S3.
                   // We will print the message for why the upload to S3 failed
from the status message.
                   // If the status was "InProgress", the status indicates that the
server has started uploading the S3 task.
                   if (Status.Success.equals(statusMessage.getStatus())) {
                       System.out.println("Successfully uploaded file at path " +
FILE_URL + " to S3.");
                       isS3UploadComplete = true;
                    } else if (Status.Failure.equals(statusMessage.getStatus()) ||
Status.Canceled.equals(statusMessage.getStatus())) {
                       System.out.println(String.format("Unable to upload file at
path %s to S3. Message %s",
```

```
 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referencia del SDK de Java: readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require(*'aws-greengrass-stream-manager-sdk'*);

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
```

```
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
            }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referencia del SDK de Node.js: readMessages | StatusMessage

Cómo realizar la inferencia de machine learning

Con él AWS IoT Greengrass, puede realizar inferencias de aprendizaje automático (ML) en sus dispositivos periféricos a partir de datos generados localmente mediante modelos entrenados en la nube. Benefíciese de la baja latencia y el ahorro de costos que supone la ejecución de inferencias locales, aprovechando al mismo tiempo la potencia de cómputo de la nube para el entrenamiento de modelos y el procesamiento complejo.

AWS IoT Greengrass hace que los pasos necesarios para realizar la inferencia sean más eficientes. Puede entrenar sus modelos de inferencia en cualquier lugar e implementarlos localmente como componentes de machine learning. Por ejemplo, puede crear y entrenar modelos de aprendizaje profundo en Amazon SageMaker AI. A continuación, puede almacenar estos modelos en un bucket de Amazon S3, de forma que pueda utilizarlos como artefactos en sus componentes para realizar inferencias en sus dispositivos principales.

Temas

- Cómo funciona la inferencia AWS IoT Greengrass de aprendizaje automático
- ¿Qué hay de diferente en la AWS IoT Greengrass versión 2?
- Requisitos
- Orígenes de modelos admitidos
- Tiempos de ejecución de machine learning compatibles
- Componentes de machine learning proporcionados por AWS
- Utilice Amazon SageMaker AI Edge Manager en los dispositivos principales de Greengrass
- Personalización de sus componentes de machine learning
- Resolución de problemas de inferencia de machine learning

Cómo funciona la inferencia AWS IoT Greengrass de aprendizaje automático

AWS proporciona componentes de aprendizaje automático que puede usar para crear implementaciones en un solo paso para realizar inferencias de aprendizaje automático en su dispositivo. También puede utilizar estos componentes como plantillas para crear componentes personalizados que se adapten a sus requisitos específicos.

AWS proporciona las siguientes categorías de componentes de aprendizaje automático:

- Componente de modelo: contiene modelos de machine learning como artefactos de Greengrass.
- Componente de tiempo de ejecución: contiene el script que instala el marco de machine learning y sus dependencias en el dispositivo principal de Greengrass.
- Componente de inferencia: contiene el código de inferencia e incluye las dependencias de los componentes para instalar el marco de machine learning y descargar modelos de machine learning previamente entrenados.

Cada implementación que cree para realizar inferencias de machine learning consta de al menos un componente que ejecuta la aplicación de inferencia, instala el marco de machine learning y descarga sus modelos de machine learning. Para realizar una inferencia de ejemplo con los componentes AWS proporcionados, debe implementar un componente de inferencia en el dispositivo principal, que incluye automáticamente los componentes del modelo y del tiempo de ejecución correspondientes como dependencias. Para personalizar las implementaciones, puede conectar o cambiar los componentes del modelo de muestra por componentes de modelo personalizados, o puede utilizar las recetas de componentes de los componentes AWS proporcionados como plantillas para crear sus propios componentes personalizados de inferencia, modelo y tiempo de ejecución.

Cómo realizar inferencias de machine learning mediante componentes personalizados:

1. Cree un componente de modelo personalizado. Este componente contiene los modelos de aprendizaje automático que desea utilizar para realizar la inferencia. AWS proporciona ejemplos de modelos DLR y TensorFlow Lite previamente entrenados. Para usar un modelo personalizado, cree su propio componente de modelo.
2. Cree un componente de tiempo de ejecución. Este componente contiene los scripts necesarios para instalar el entorno de ejecución de aprendizaje automático en sus modelos. AWS proporciona ejemplos de componentes de tiempo de ejecución para Deep Learning Runtime (DLR) y TensorFlow Lite. Para usar otros tiempos de ejecución con sus modelos personalizados y su código de inferencia, cree sus propios componentes de tiempo de ejecución.
3. Cree un componente de inferencia. Este componente contiene el código de inferencia e incluye los componentes del modelo y del tiempo de ejecución como dependencias. AWS proporciona ejemplos de componentes de inferencia para la clasificación de imágenes y la detección de objetos mediante DLR y Lite. TensorFlow Para realizar otros tipos de inferencias, o para usar modelos y tiempos de ejecución personalizados, cree su propio componente de inferencia.

4. Implemente el componente de inferencia. Al implementar este componente, AWS IoT Greengrass también despliega automáticamente las dependencias del modelo y del componente de tiempo de ejecución.

Para empezar con los componentes AWS proporcionados, consulte. the section called “Realización de una inferencia de clasificación de imágenes de muestra”

Para obtener información sobre cómo crear componentes de machine learning personalizados, consulte Personalización de sus componentes de machine learning.

¿Qué hay de diferente en la AWS IoT Greengrass versión 2?

AWS IoT Greengrass consolida las unidades funcionales del aprendizaje automático (como los modelos, los tiempos de ejecución y el código de inferencia) en componentes que permiten utilizar un proceso de un solo paso para instalar el entorno de ejecución del aprendizaje automático, descargar los modelos entrenados y realizar inferencias en el dispositivo.

Al utilizar los componentes de aprendizaje automático AWS proporcionados, tiene la flexibilidad de empezar a realizar inferencias de aprendizaje automático con ejemplos de códigos de inferencia y modelos previamente entrenados. Puede conectar componentes de modelos personalizados para utilizar sus propios modelos personalizados con los componentes de inferencia y tiempo de ejecución que proporcionan. AWS Para obtener una solución de machine learning completamente personalizada, puede usar los componentes públicos como plantillas para crear componentes personalizados y usar cualquier tiempo de ejecución, modelo o tipo de inferencia que desee.

Requisitos

Para crear y utilizar componentes de machine learning, debe disponer de lo siguiente:

- Un dispositivo principal de Greengrass. Si no dispone de una, consulte Tutorial: Introducción a AWS IoT Greengrass V2.
- Espacio de almacenamiento local mínimo de 500 MB para usar los componentes de aprendizaje AWS automático de muestra proporcionados.

Orígenes de modelos admitidos

AWS IoT Greengrass admite el uso de modelos de aprendizaje automático personalizados que se almacenan en Amazon S3. También puede utilizar los trabajos de empaquetado perimetral de Amazon SageMaker AI para crear directamente componentes de modelo para sus modelos compilados con SageMaker AI NEO. Para obtener información sobre el uso de SageMaker AI Edge Manager con AWS IoT Greengrass, consulteUtilice Amazon SageMaker AI Edge Manager en los dispositivos principales de Greengrass.

Los buckets de S3 que contienen sus modelos deben cumplir los siguientes requisitos:

- No deben cifrarse con SSE-C. En el caso de los buckets que utilizan cifrado del lado del servidor, la inferencia de machine learning de AWS IoT Greengrass actualmente solo admite las opciones de cifrado SSE-S3 o SSE-KMS. Para obtener más información sobre las opciones de cifrado del lado del servidor, consulte Protección de datos con el cifrado del lado del servidor en la Guía del usuario de Amazon Simple Storage Service.
- Sus nombres no deben incluir puntos (`.`). Para obtener más información, consulte la regla sobre el uso de buckets virtuales de estilo host con SSL en Reglas para la nomenclatura del bucket en la Guía del usuario de Amazon Simple Storage Service.
- Los depósitos de S3 que almacenan las fuentes de los modelos deben estar en los mismos componentes de aprendizaje automático Cuenta de AWS y Región de AWS al igual que ellos.
- AWS IoT Greengrass debe tener `read` permiso para acceder a la fuente del modelo. Para permitir el acceso AWS IoT Greengrass a los buckets de S3, el rol de dispositivo de Greengrass debe permitir `s3:GetObject` la acción. Para obtener más información acerca del rol del dispositivo, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Tiempos de ejecución de machine learning compatibles

AWS IoT Greengrass le permite crear componentes personalizados para utilizar cualquier entorno de aprendizaje automático que elija para realizar inferencias de aprendizaje automático con sus modelos personalizados. Para obtener información sobre cómo crear componentes de machine learning personalizados, consulte Personalización de sus componentes de machine learning.

Para que el proceso de introducción al aprendizaje automático sea más eficiente, AWS IoT Greengrass proporciona ejemplos de componentes de inferencia, modelos y tiempo de ejecución que utilizan los siguientes tiempos de ejecución de aprendizaje automático:

- Tiempo de ejecución de aprendizaje profundo (DLR) versión 1.6.0 y versión 1.3.0
- TensorFlow Lite v2.5.0

Componentes de machine learning proporcionados por AWS

En la siguiente tabla se enumeran los componentes AWS proporcionados que se utilizan para el aprendizaje automático.

> **Note**
>
> Varios AWS de los componentes proporcionados dependen de versiones secundarias específicas del núcleo de Greengrass. Debido a esta dependencia, es necesario actualizar estos componentes al actualizar el núcleo de Greengrass a una nueva versión secundaria. Para obtener información sobre las versiones específicas del núcleo de las que depende cada componente, consulte el tema del componente correspondiente. Para más información sobre la actualización del núcleo, consulte Actualización del software AWS IoT Greengrass Core (OTA).

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
SageMaker Administrador AI Edge	Implementa el agente Amazon SageMaker AI Edge Manager en el dispositivo principal de Greengrass.	Genérico	Linux, Windows	No
Clasificación de imágenes de DLR	Componente de inferencia que utiliza el almacén de	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
	modelos de clasificación de imágenes de DLR y el componente de tiempo de ejecución de DLR como dependencias para instalar el DLR, descargar modelos de clasificación de imágenes de muestra y realizar inferencias de clasificación de imágenes en los dispositivos compatibles.			

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
Detección de objetos del DLR	Componente de inferencia que utiliza el almacén de modelos de detección de objetos del DLR y el componente de tiempo de ejecución del DLR como dependencias para instalar el DLR, descargar modelos de detección de objetos de muestra y realizar inferencias de detección de objetos en los dispositivos compatibles.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
Almacén de modelos de clasificación de imágenes de DLR	Componente de modelo que contiene ejemplos de ResNet -50 modelos de clasificación de imágenes como artefactos de Greengrass.	Genérico	Linux, Windows	No
Almacén de modelos de detección de objetos del DLR	Componente de modelo que contiene ejemplos de modelos de detección de YOLOv3 objetos como artefactos de Greengrass.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
Tiempo de ejecución de DLR	Componente de tiempo de ejecución que contiene una cadena de instalación que se utiliza para instalar el DLR y sus dependencias en el dispositivo principal de Greengrass.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Clasificación de imágenes Lite	Componente de inferencia que utiliza el TensorFlow almacén de modelos de clasificación de imágenes de TensorFlow Lite y el componente de tiempo de ejecución de Lite como dependencias para instalar TensorFlow Lite, descargar modelos de clasificación de imágenes de muestra y realizar inferencias de clasificación de imágenes en dispositivos compatibles.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Detección de objetos Lite	Componente de inferencia que utiliza la tienda de modelos de detección de objetos de TensorFlow Lite y el componente de tiempo de ejecución de TensorFlow Lite como dependencias para instalar TensorFlow Lite, descargar modelos de detección de objetos de muestra y realizar inferencias de detección de objetos en dispositivos compatibles.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Tienda de modelos de clasificación de imágenes Lite	Componente de modelo que contiene un modelo MobileNet v1 de muestra como artefacto de Greengrass.	Genérico	Linux, Windows	No
TensorFlow Tienda de modelos de detección de objetos Lite	Componente de modelo que contiene un MobileNet modelo de muestra de detección de disparo único (SSD) como un artefacto de Greengrass.	Genérico	Linux, Windows	No

Componente	Description (Descripción)	Tipo de componente	Sistema operativo admitido	Código abierto
TensorFlow Tiempo de ejecución Lite	Componente de tiempo de ejecución que contiene un script de instalación que se utiliza para instalar TensorFlow Lite y sus dependencias en el dispositivo principal de Greengrass.	Genérico	Linux, Windows	No

Utilice Amazon SageMaker AI Edge Manager en los dispositivos principales de Greengrass

> **Important**
>
> SageMaker AI Edge Manager dejó de fabricarse el 26 de abril de 2024. Para obtener más información sobre cómo seguir implementando sus modelos en dispositivos periféricos, consulte el final del ciclo de vida de SageMaker AI Edge Manager.

Amazon SageMaker AI Edge Manager es un agente de software que se ejecuta en dispositivos periféricos. SageMaker AI Edge Manager proporciona administración de modelos para dispositivos periféricos para que pueda empaquetar y usar modelos compilados por Amazon SageMaker AI NEO directamente en los dispositivos principales de Greengrass. Con SageMaker AI Edge Manager, también puede muestrear los datos de entrada y salida del modelo de sus dispositivos principales y enviarlos a ellos Nube de AWS para su supervisión y análisis. Como SageMaker AI Edge Manager

utiliza SageMaker AI Neo para optimizar tus modelos para el hardware de destino, no necesitas instalar el DLR Runtime directamente en tu dispositivo. En los dispositivos Greengrass, SageMaker AI Edge Manager no carga los AWS IoT certificados locales ni llama directamente al punto final del proveedor de AWS IoT credenciales. En su lugar, SageMaker AI Edge Manager utiliza el servicio de intercambio de fichas para obtener una credencial temporal de un punto final de TES.

En esta sección se describe cómo funciona SageMaker AI Edge Manager en los dispositivos principales de Greengrass.

Cómo funciona SageMaker AI Edge Manager en los dispositivos Greengrass

Para implementar el agente SageMaker AI Edge Manager en sus dispositivos principales, cree una implementación que incluya el `aws.greengrass.SageMakerEdgeManager` componente. AWS IoT Greengrass gestiona la instalación y el ciclo de vida del agente Edge Manager en sus dispositivos. Cuando haya disponible una nueva versión del binario del agente, implemente la versión actualizada del componente `aws.greengrass.SageMakerEdgeManager` para actualizar la versión del agente que está instalada en su dispositivo.

Cuando utilizas SageMaker AI Edge Manager con AWS IoT Greengrass, tu flujo de trabajo incluye los siguientes pasos de alto nivel:

1. Compila modelos con SageMaker AI Neo.
2. Empaque sus modelos compilados con SageMaker AI Neo utilizando trabajos de empaquetado periféricos de SageMaker AI. Cuando ejecuta un trabajo de empaquetado de periferia para su modelo, puede optar por crear un componente del modelo con el modelo empaquetado como un artefacto que se puede implementar en su dispositivo principal de Greengrass.
3. Cree un componente de inferencia personalizado. Este componente de inferencia se utiliza para interactuar con el agente del administrador de periféricos y realizar la inferencia en el dispositivo principal. Estas operaciones incluyen cargar modelos, invocar solicitudes de predicción para ejecutar la inferencia y descargar modelos cuando el componente se apaga.
4. Implemente el componente SageMaker AI Edge Manager, el componente de modelo empaquetado y el componente de inferencia para ejecutar el modelo en el motor de inferencia de SageMaker IA (agente Edge Manager) de su dispositivo.

Para obtener más información sobre la creación de trabajos de empaquetado perimetral y componentes de inferencia que funcionen con SageMaker AI Edge Manager, consulte Deploy Model

Package y Edge Manager Agent con AWS IoT Greengrass en la Guía para desarrolladores de Amazon SageMaker AI.

El Tutorial: Cómo empezar a SageMaker usar AI Edge Manager tutorial le muestra cómo configurar y usar el agente SageMaker AI Edge Manager en un dispositivo principal de Greengrass existente, utilizando el código de ejemplo AWS proporcionado que puede usar para crear ejemplos de componentes de inferencia y modelo.

Cuando usa SageMaker AI Edge Manager en los dispositivos principales de Greengrass, también puede usar la función de captura de datos para cargar datos de muestra en. Nube de AWS La captura de datos es una función de SageMaker IA que se utiliza para cargar entradas de inferencia, resultados de inferencias y datos de inferencia adicionales a un bucket de S3 o a un directorio local para futuros análisis. Para obtener más información sobre el uso de datos de captura con SageMaker AI Edge Manager, consulte Manage Model en la Guía para desarrolladores de Amazon SageMaker AI.

Requisitos

Debe cumplir los siguientes requisitos para utilizar el agente SageMaker AI Edge Manager en los dispositivos principales de Greengrass.

- Un dispositivo principal de Greengrass que se ejecuta en Amazon Linux 2, una plataforma de Linux basada en Debian (x86_64 o Armv8) o Windows (x86_64). Si no dispone de una, consulte Tutorial: Introducción a AWS IoT Greengrass V2.
- Python 3.6 o posterior, incluido `pip` para la versión de Python, instalado en el dispositivo principal.
- El rol del dispositivo de Greengrass se configuró con lo siguiente:
  - Una relación de confianza que permite a `credentials.iot.amazonaws.com` y a `sagemaker.amazonaws.com` asumir el rol, como se muestra en el siguiente ejemplo de política de IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
```

```
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- La política gestionada AmazonSageMakerEdgeDeviceFleetPolicyde IAM.
- La acción `s3:PutObject`, como se muestra en el siguiente ejemplo de política de IAM.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Un bucket de Amazon S3 creado en el mismo dispositivo principal de Greengrass Cuenta de AWS y Región de AWS en el mismo que él. SageMaker AI Edge Manager requiere un depósito S3 para crear una flota de dispositivos perimetrales y almacenar datos de muestra derivados de la ejecución de inferencias en su dispositivo. Para más información sobre la creación de buckets de S3, consulte Introducción a Amazon S3.
- Una flota de dispositivos periféricos de SageMaker IA que utiliza el mismo alias de AWS IoT rol que su dispositivo principal de Greengrass. Para obtener más información, consulte Creación de una flota de dispositivos de periferia.
- Su dispositivo principal de Greengrass registrado como dispositivo perimetral en su flota de dispositivos SageMaker AI Edge. El nombre del dispositivo perimetral debe coincidir con el AWS IoT nombre del dispositivo principal. Para obtener más información, consulte Registro del dispositivo principal de Greengrass.

Comience con SageMaker AI Edge Manager

Puedes completar un tutorial para empezar a usar SageMaker AI Edge Manager. El tutorial muestra cómo empezar a utilizar SageMaker AI Edge Manager con los componentes AWS de muestra proporcionados en un dispositivo principal existente. Estos componentes de ejemplo utilizan el componente SageMaker AI Edge Manager como una dependencia para implementar el agente de Edge Manager y realizar inferencias utilizando modelos previamente entrenados que se compilaron con SageMaker AI Neo. Para obtener más información, consulte Tutorial: Cómo empezar a SageMaker usar AI Edge Manager.

Personalización de sus componentes de machine learning

En AWS IoT Greengrass, puede configurar ejemplos de componentes de aprendizaje automático para personalizar la forma en que realiza la inferencia de aprendizaje automático en sus dispositivos, con los componentes de inferencia, modelo y tiempo de ejecución como componentes básicos. AWS IoT Greengrass también le proporciona la flexibilidad de usar los componentes de muestra como plantillas y crear sus propios componentes personalizados según sea necesario. Puede combinar este enfoque modular para personalizar los componentes de inferencia del machine learning de las siguientes maneras:

Uso de ejemplos de componentes de inferencia

- Modifique la configuración de los componentes de inferencia cuando los implemente.
- Utilice un modelo personalizado con el componente de inferencia de muestra sustituyendo el componente almacén de modelos de muestra por un componente de modelo personalizado. El modelo personalizado debe entrenarse con el mismo tiempo de ejecución que el modelo de muestra.

Uso de componentes de inferencia personalizados

- Use código de inferencia personalizado con los modelos y tiempos de ejecución de muestra agregando componentes de modelos públicos y componentes de tiempo de ejecución como dependencias de los componentes de inferencia personalizados.
- Cree y agrega componentes de modelo personalizados o componentes de tiempo de ejecución como dependencias de componentes de inferencia personalizados. Debe utilizar componentes personalizados si desea utilizar un código de inferencia personalizado o un entorno de ejecución para el que AWS IoT Greengrass no se proporcione un componente de muestra.

Temas

- Modificación de la configuración de un componente de inferencia público
- Uso de un modelo personalizado con el componente de inferencia de muestra
- Creación de componentes de machine learning personalizados
- Creación de un componente de inferencia personalizado

Modificación de la configuración de un componente de inferencia público

En la consola de AWS IoT Greengrass, la página del componente muestra la configuración predeterminada de ese componente. Por ejemplo, la configuración predeterminada del componente de clasificación de imágenes de TensorFlow Lite es la siguiente:

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
}
```

Cuando implementa un componente de inferencia público, puede modificar la configuración predeterminada para personalizar su implementación. Para obtener información sobre los parámetros de configuración disponibles para cada componente de inferencia pública, consulte el tema del componente en Componentes de machine learning proporcionados por AWS.

En esta sección se describe cómo implementar un componente modificado desde la AWS IoT Greengrass consola. Para obtener información sobre la implementación de componentes mediante el AWS CLI, consulteCrear implementaciones.

Cómo implementar un componente de inferencia pública modificado (consola)

1. Inicie sesión en la consola de AWS IoT Greengrass.
2. En el menú de navegación, elija Componentes.
3. En la página Componentes, en la pestaña Componentes públicos, elija el componente que desea implementar.
4. En la página del componente, elija Implementar.
5. En Agregar a la implementación, elija una de las siguientes opciones:
   a. Para combinar este componente con una implementación existente en el dispositivo de destino, elija Agregar a la implementación existente y, a continuación, seleccione la implementación que desee revisar.
   b. Para crear una nueva implementación en el dispositivo de destino, elija Crear nueva implementación. Si tiene una implementación existente en su dispositivo, al elegir este paso se reemplaza la implementación existente.
6. En la página Especificar detalles, haga lo siguiente:
   a. En Información de implementación, introduzca o modifique el nombre descriptivo de su implementación.
   b. En Objetivos de implementación, seleccione un objetivo para su implementación y elija Siguiente. No puede cambiar el objetivo de implementación si está revisando una implementación existente.
7. En la página Seleccionar componentes, en Componentes públicos, compruebe que está seleccionado el componente de inferencia con la configuración modificada y elija Siguiente.
8. En la página Configurar componentes, haga lo siguiente:
   a. Seleccione el componente de inferencia y elija Configurar componente.
   b. En Actualización de configuración, introduzca los valores de configuración que desee actualizar. Por ejemplo, introduzca la siguiente actualización de configuración en el cuadro Configuración para combinar para cambiar el intervalo de inferencia a 15 segundos e indique al componente que busque la imagen con nombre `custom.jpg` en la carpeta `/custom-ml-inference/images/`.

```
{
  "InferenceInterval": "15",
  "ImageName": "custom.jpg",
  "ImageDirectory": "/custom-ml-inference/images/"
}
```

Para restablecer toda la configuración de un componente a sus valores predeterminados, especifique una sola cadena vacía `""` en el cuadro Restablecer rutas.

c. Seleccione Confirmar y, a continuación, elija Siguiente.

9. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
10. En la página Revisar, elija Implementar.

Uso de un modelo personalizado con el componente de inferencia de muestra

Si desea utilizar el componente de inferencia de ejemplo con sus propios modelos de aprendizaje automático para un tiempo de ejecución que AWS IoT Greengrass proporcione un componente de tiempo de ejecución de ejemplo, debe anular los componentes del modelo público por componentes que utilicen esos modelos como artefactos. En general, debe completar los siguientes pasos para usar un modelo personalizado con el componente de inferencia de muestra:

1. Cree un componente de modelo que utilice un modelo personalizado en un bucket de S3 como artefacto. Su modelo personalizado debe entrenarse con el mismo tiempo de ejecución que el modelo que desea reemplazar.
2. Modifique el parámetro de configuración `ModelResourceKey` en el componente de inferencia para usar el modelo personalizado. Para obtener información sobre la actualización de la configuración del componente de inferencia, consulte Modificación de la configuración de un componente de inferencia público.

Al implementar el componente de inferencia, AWS IoT Greengrass busca la versión más reciente de las dependencias de sus componentes. Anula el componente del modelo público dependiente si existe una versión personalizada posterior del componente en la misma banda. Cuenta de AWS Región de AWS

Creación de un componente de modelo personalizado (consola)

1. Suba un modelo en su bucket de S3. Para obtener más información acerca de cargar sus modelos a un bucket de S3, consulte Cómo trabajar con buckets de Amazon S3 en la Guía del usuario de Amazon Simple Storage Service.

   > **Note**
   >
   > Debe almacenar sus artefactos en depósitos de S3 que estén en los mismos Cuenta de AWS y Región de AWS como los componentes. Para permitir el acceso AWS IoT Greengrass a estos artefactos, el rol del dispositivo Greengrass debe permitir la `s3:GetObject` acción. Para obtener más información acerca del rol del dispositivo, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

2. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
3. Recupere la receta del componente para el componente del almacén de modelos público.

   a. En la página Componentes, en la pestaña Componentes públicos, busque y elija el componente del modelo público para el que desee crear una nueva versión. Por ejemplo, `variant.DLR.ImageClassification.ModelStore`.

   b. En la página de componentes, elija Ver receta y copie la receta JSON que se muestra.
4. En en la página Componentes, en la pestaña Mis componentes, elija Crear componente.
5. En la página Crear componente, en Información del componente, seleccione Introducir la receta como JSON como origen del componente.
6. En el cuadro Receta, pegue la receta del componente que copió anteriormente.
7. En la receta, actualice los siguientes valores:

   - `ComponentVersion`: aumenta la versión menor del componente.

     Cuando cree un componente personalizado para anular un componente un modelo público, debe actualizar solo la versión menor de la versión del componente existente. Por ejemplo, si la versión del componente público es `2.1.0`, puede crear un componente personalizado con la versión `2.1.1`.

   - `Manifests.Artifacts.Uri`: actualiza cada valor de URI al URI de Amazon S3 del modelo que desea utilizar.

> **Note**
>
> No cambie el nombre del componente.

8. Seleccione Crear componente.

Creación de un componente de modelo personalizado (AWS CLI)

1. Suba un modelo en su bucket de S3. Para obtener más información acerca de cargar sus modelos a un bucket de S3, consulte Cómo trabajar con buckets de Amazon S3 en la Guía del usuario de Amazon Simple Storage Service.

   > **Note**
   >
   > Debe almacenar sus artefactos en depósitos S3 que estén en el mismo lugar que Cuenta de AWS los Región de AWS componentes. Para permitir el acceso AWS IoT Greengrass a estos artefactos, el rol del dispositivo Greengrass debe permitir la `s3:GetObject` acción. Para obtener más información acerca del rol del dispositivo, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

2. Ejecute el siguiente comando para recuperar la receta de componente del componente público. Este comando escribe la receta del componente en el archivo de salida que usted proporcione en el comando. Convierta la cadena codificada base64 recuperada a JSON o YAML, según sea necesario.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn <arn> \
       --recipe-output-format <recipe-format> \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn <arn> ^
   ```

```
    --recipe-output-format <recipe-format> ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

3. Actualice el nombre del archivo de recetas a *<component-name>*-*<component-version>*, donde la versión del componente sea la versión de destino del nuevo componente. Por ejemplo, `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`.
4. En la receta, actualice los siguientes valores:

   - `ComponentVersion`: aumenta la versión menor del componente.

     Cuando cree un componente personalizado para anular un componente un modelo público, debe actualizar solo la versión menor de la versión del componente existente. Por ejemplo, si la versión del componente público es `2.1.0`, puede crear un componente personalizado con la versión `2.1.1`.
   - `Manifests.Artifacts.Uri`: actualiza cada valor de URI al URI de Amazon S3 del modelo que desea utilizar.

   > **Note**
   >
   > No cambie el nombre del componente.

5. Ejecute el siguiente comando para crear un componente nuevo con la receta que recuperó y modificó.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/component/recipe
```

> **Note**
>
> Este paso crea el componente del AWS IoT Greengrass servicio en. Nube de AWS Puede utilizar la CLI de Greengrass para desarrollar, probar e implementar su componente de forma local antes de cargarlo en la nube. Para obtener más información, consulte Desarrollo de componentes de AWS IoT Greengrass.

Para obtener más información sobre cómo crear componentes, consulte Desarrollo de componentes de AWS IoT Greengrass.

Creación de componentes de machine learning personalizados

Debe crear componentes personalizados si desea utilizar un código de inferencia personalizado o un tiempo de ejecución para el que AWS IoT Greengrass no se proporcione un componente de muestra. Puede usar su código de inferencia personalizado con los modelos de aprendizaje automático y tiempos de ejecución de muestra AWS proporcionados, o puede desarrollar una solución de inferencia de aprendizaje automático completamente personalizada con sus propios modelos y tiempo de ejecución. Si sus modelos utilizan un entorno de ejecución para el que se AWS IoT Greengrass proporciona un ejemplo de componente de tiempo de ejecución, puede utilizar ese componente de tiempo de ejecución y tendrá que crear componentes personalizados únicamente para el código de inferencia y los modelos que desee utilizar.

Temas

- Obtención de la receta de un componente público
- Obtención de artefactos de componentes de muestra
- Carga de artefactos de componentes en un bucket de S3
- Creación de componentes personalizados

Obtención de la receta de un componente público

Puede utilizar la receta de un componente público de machine learning existente como plantilla para crear un componente personalizado. Para ver la receta de componentes de la última versión de un componente público, utilice la consola o utilice la AWS CLI siguiente opción:

- Uso de la consola

1. En la página Componentes, en la pestaña Componentes públicos, busque y elija el componente público.
2. En la página del componente, elija Ver receta.

- Usando AWS CLI

  Ejecute el siguiente comando para recuperar la receta de componente del componente de la variante pública. Este comando escribe la receta del componente en el archivo de receta JSON o YAML que proporcione en el comando.

  Linux, macOS, or Unix

  ```
  aws greengrassv2 get-component \
      --arn <arn> \
      --recipe-output-format <recipe-format> \
      --query recipe \
      --output text | base64 --decode > <recipe-file>
  ```

  Windows Command Prompt (CMD)

  ```
  aws greengrassv2 get-component ^
      --arn <arn> ^
      --recipe-output-format <recipe-format> ^
      --query recipe ^
      --output text > <recipe-file>.base64

  certutil -decode <recipe-file>.base64 <recipe-file>
  ```

  PowerShell

  ```
  aws greengrassv2 get-component `
      --arn <arn> `
      --recipe-output-format <recipe-format> `
      --query recipe `
      --output text > <recipe-file>.base64

  certutil -decode <recipe-file>.base64 <recipe-file>
  ```

  Sustituya los valores del comando de la siguiente manera:

  - *<arn>*. El Nombre de recurso de Amazon (ARN) del componente público.

- *<recipe-format>*. El formato en el que desea crear el archivo de recetas. Los valores admitidos son `JSON` y `YAML`.
- *<recipe-file>*. El nombre de la receta en el formato *<component-name>*-*<component-version>*.

Obtención de artefactos de componentes de muestra

Puede utilizar los artefactos que utilizan los componentes públicos de machine learning como plantillas para crear sus artefactos de componentes personalizados, como códigos de inferencia o cadenas de instalación en tiempo de ejecución.

Para ver los artefactos de muestra que se incluyen en los componentes públicos de machine learning, implemente el componente de inferencia público y, a continuación, visualice los artefactos del dispositivo en la carpeta */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/`*component-version*`/`.

Carga de artefactos de componentes en un bucket de S3

Antes de poder crear un componente personalizado, debe cargar los artefactos del componente en un bucket de S3 y utilizar el S3 URIs en la receta del componente. Por ejemplo, para usar un código de inferencia personalizado en su componente de inferencia, cargue el código en un bucket de S3. A continuación, puede utilizar el URI de Amazon S3 de su código de inferencia como un artefacto en su componente.

Para obtener más información acerca de cargar contenido a un bucket de S3, consulte Cómo trabajar con buckets de Amazon S3 en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> Debe almacenar los artefactos en depósitos de S3 que estén en el mismo lugar que Cuenta de AWS Región de AWS los componentes. Para permitir el acceso AWS IoT Greengrass a estos artefactos, el rol del dispositivo Greengrass debe permitir la `s3:GetObject` acción. Para obtener más información acerca del rol del dispositivo, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Creación de componentes personalizados

Puede utilizar los artefactos y las recetas que haya recuperado para crear sus componentes de machine learning personalizados. Para ver un ejemplo, consulta Creación de un componente de inferencia personalizado.

Para obtener información detallada sobre la creación e implementación de componentes en los dispositivos de Greengrass, consulte Desarrollo de componentes de AWS IoT Greengrass y Implemente AWS IoT Greengrass componentes en los dispositivos.

Creación de un componente de inferencia personalizado

En esta sección, se muestra cómo crear un componente de inferencia personalizado con el componente de clasificación de imágenes de DLR como una plantilla.

Temas

- Carga de su código de inferencia a un bucket de Amazon S3
- Creación de una receta para su componente de inferencia
- Creación del componente de inferencia

Carga de su código de inferencia a un bucket de Amazon S3

Cree su código de inferencia y, a continuación, cárguelo en un bucket de S3. Para obtener más información acerca de cargar contenido a un bucket de S3, consulte Cómo trabajar con buckets de Amazon S3 en la Guía del usuario de Amazon Simple Storage Service.

> **Note**
>
> Debe almacenar sus artefactos en depósitos S3 que estén en el mismo lugar que Cuenta de AWS los Región de AWS componentes. Para permitir el acceso AWS IoT Greengrass a estos artefactos, el rol del dispositivo Greengrass debe permitir la `s3:GetObject` acción. Para obtener más información acerca del rol del dispositivo, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Creación de una receta para su componente de inferencia

1. Ejecute el siguiente comando para recuperar la receta de componente del componente de clasificación de imágenes de DLR. Este comando escribe la receta del componente en el archivo de receta JSON o YAML que proporcione en el comando.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    \
       --recipe-output-format JSON | YAML \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    ^
       --recipe-output-format JSON | YAML ^
       --query recipe ^
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

   PowerShell

   ```
   aws greengrassv2 get-component `
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    `
       --recipe-output-format JSON | YAML `
       --query recipe `
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

*<recipe-file>*Sustitúyalo por el nombre de la receta en el formato*<component-name>*-*<component-version>*.

2. En el objeto `ComponentDependencies` de la receta, realice una o varias de las siguientes acciones según el modelo y los componentes del tiempo de ejecución que desee utilizar:

   - Mantenga la dependencia de los componentes del DLR si desea utilizar modelos compilados por el DLR. También puede reemplazarlo por una dependencia de un componente de tiempo de ejecución personalizado, como se muestra en el ejemplo siguiente.

     Componente de tiempo de ejecución

     JSON

     ```
     {
         "<runtime-component>": {
             "VersionRequirement": "<version>",
             "DependencyType": "HARD"
         }
     }
     ```

     YAML

     ```
     <runtime-component>:
         VersionRequirement: "<version>"
         DependencyType: HARD
     ```

   - Mantenga la dependencia del almacén de modelos de clasificación de imágenes del DLR para utilizar los modelos ResNet -50 previamente entrenados que AWS proporciona, o modifíquelo para utilizar un componente de modelo personalizado. Al incluir una dependencia para un componente de modelo público, si existe una versión personalizada posterior del componente en el mismo Cuenta de AWS y Región de AWS, entonces, el componente de inferencia utiliza ese componente personalizado. Especifique la dependencia del componente del modelo como se muestra en los siguientes ejemplos.

     Componente de modelo público

     JSON

     ```
     {
         "variant.DLR.ImageClassification.ModelStore": {
     ```

```
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
variant.DLR.ImageClassification.ModelStore:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

Componente de modelo personalizado

JSON

```
{
    "<custom-model-component>": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. En el objeto `ComponentConfiguration`, agregue la configuración por defecto para este componente. Más adelante podrá modificar esta configuración cuando implemente el componente. El siguiente extracto muestra la configuración del componente de clasificación de imágenes de DLR.

   Por ejemplo, si usa un componente de modelo personalizado como dependencia para su componente de inferencia personalizado, modifique `ModelResourceKey` para proporcionar los nombres de los modelos que está utilizando.

   JSON

```
{
  "accessControl": {
```

```
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
    aws.greengrass.ipc.mqttproxy:
        'aws.greengrass.ImageClassification:mqttproxy:1':
            policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
            operations:
                - 'aws.greengrass#PublishToIoTCore'
            resources:
                - ml/dlr/image-classification
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
    armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
    x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
    aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. En el objeto `Manifests`, proporcione información sobre los artefactos y la configuración de este componente que se utilizan cuando el componente se implementa en diferentes plataformas y cualquier otra información necesaria para ejecutar correctamente el componente. El siguiente extracto muestra la configuración del objeto `Manifests` para la plataforma Linux en el componente de clasificación de imágenes de DLR.

   JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
          "DLR_IC_MODEL_DIR":
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "Run": {
          "RequiresPrivilege": true,
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

YAML

```
Manifests:
  - Platform:
      os: linux
      architecture: arm
    Name: 32-bit armv7l - Linux (raspberry pi)
    Artifacts:
      - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip
        Unarchive: ZIP
    Lifecycle:
      SetEnv:
        DLR_IC_MODEL_DIR:
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}"
        DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/
image_classification/sample_images/"
      Run:
        RequiresPrivilege: true
        script: |-
          . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/
activate
          python3 {artifacts:decompressedPath}/image_classification/inference.py
```

Para obtener más información acerca de la creación de recetas de componentes, consulte AWS IoT Greengrass referencia de recetas de componentes.

Creación del componente de inferencia

Utilice la AWS IoT Greengrass consola o el AWS CLI para crear un componente con la receta que acaba de definir. Una vez creado el componente, puede implementarlo para realizar inferencias en su dispositivo. Para ver un ejemplo de cómo implementar un componente de inferencia, consulte Tutorial: Realice una inferencia de clasificación de imágenes de muestra con Lite TensorFlow.

Creación de un componente de inferencia personalizado (consola)

1. Inicie sesión en la consola de AWS IoT Greengrass.
2. En el menú de navegación, elija Componentes.
3. En en la página Componentes, en la pestaña Mis componentes, elija Crear componente.

4. En la página Crear componente, en Información del componente, seleccione Introducir la receta como JSON o Introducir la receta como YAML como origen del componente.
5. En el cuadro Receta, introduzca la receta personalizada que haya creado.
6. Haga clic en Crear componente.

Creación de un componente de inferencia personalizado (AWS CLI)

Ejecute el siguiente comando para crear un componente personalizado nuevo con la receta que creó.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/recipe/file
```

> **Note**
>
> Este paso crea el componente del AWS IoT Greengrass servicio en Nube de AWS. Puede utilizar la CLI de Greengrass para desarrollar, probar e implementar su componente de forma local antes de cargarlo en la nube. Para obtener más información, consulte Desarrollo de componentes de AWS IoT Greengrass.

Resolución de problemas de inferencia de machine learning

Use la información de solución de problemas y las soluciones de esta sección para resolver problemas con sus componentes de machine learning. Para ver los componentes públicos de inferencia de machine learning, consulte los mensajes de error en los siguientes registros de componentes:

Linux or Unix

- */greengrass/v2*/logs/aws.greengrass.DLRImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.DLRObjectDetection.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteObjectDetection.log

Windows

- *C:\greengrass\v2*\logs\aws.greengrass.DLRImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.DLRObjectDetection.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteObjectDetection.log

Si un componente está instalado correctamente, el registro del componente contiene la ubicación de la biblioteca que utiliza para la inferencia.

Problemas

- No fue posible recuperar la biblioteca
- Cannot open shared object file
- Error: ModuleNotFoundError: No module named '<library>'
- No se ha detectado ningún dispositivo compatible con CUDA
- No existe tal archivo o directorio
- RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>
- picamera.exc.PiCameraError: Camera is not enabled
- Errores de memoria
- Errores de espacio en el disco
- Errores de tiempo de espera

No fue posible recuperar la biblioteca

El siguiente error se produce cuando la cadena de instalación no puede descargar una biblioteca requerida durante la implementación en un dispositivo Raspberry Pi.

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
 3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

Ejecute `sudo apt-get update` y vuelva a implementar el componente.

Cannot open shared object file

Es posible que se produzcan errores similares a los siguientes cuando la cadena del instalador no pueda descargar una dependencia requerida para `opencv-python` durante la implementación en un dispositivo Raspberry Pi.

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

Ejecute el siguiente comando para instalar manualmente las dependencias para `opencv-python`:

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

Es posible que vea este error en los registros de los componentes de tiempo de ejecución de ML (`variant.DLR.log` o `variant.TensorFlowLite.log`) cuando el tiempo de ejecución de la biblioteca de ML o sus dependencias no estén instaladas correctamente. Este error se puede producir en los siguientes escenarios:

- Si usa la opción `UseInstaller`, que está habilitada de forma predeterminada, este error indica que el componente de tiempo de ejecución de ML no pudo instalar el tiempo de ejecución o sus dependencias. Haga lo siguiente:
  1. Configure el componente de tiempo de ejecución de ML para deshabilitar la opción `UseInstaller`.
  2. Instale el tiempo de ejecución de ML y sus dependencias y póngalos a disposición del usuario del sistema que ejecuta los componentes de ML. Para obtener más información, consulte los siguientes temas:
     - Opción de tiempo de ejecución de DLR UseInstaller
     - TensorFlowOpción de tiempo de ejecución UseInstaller Lite
- Si no usa la opción `UseInstaller`, este error indica que el tiempo de ejecución de ML o sus dependencias no están instalados para el usuario del sistema que ejecuta los componentes de ML. Haga lo siguiente:
  1. Compruebe que la biblioteca esté instalada para el usuario del sistema que ejecuta los componentes de ML. *`ggc_user`*Sustitúyalo por el nombre del usuario del sistema y *`tflite_runtime`* sustitúyalo por el nombre de la biblioteca que desee comprobar.

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
```

Windows

```
runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
```

2. Si la biblioteca no está instalada, instálela para ese usuario. *ggc_user*Sustitúyalo por el nombre del usuario del sistema y *tflite_runtime* sustitúyalo por el nombre de la biblioteca.

   Linux or Unix

   ```
   sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
   ```

   Windows

   ```
   runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
   ```

   Para obtener más información acerca de las dependencias de cada tiempo de ejecución de ML, consulte lo siguiente:

   - Opción de tiempo de ejecución UseInstaller de DLR
   - TensorFlowOpción de tiempo de ejecución UseInstaller Lite

3. Si el problema persiste, instale la biblioteca para que otro usuario confirme si este dispositivo puede instalarla. El usuario puede ser, por ejemplo, su usuario, el usuario raíz o un usuario administrador. Si no puede instalar la biblioteca correctamente para ningún usuario, es posible que su dispositivo no sea compatible con la biblioteca. Consulte la documentación de la biblioteca para revisar los requisitos y solucionar los problemas de instalación.

No se ha detectado ningún dispositivo compatible con CUDA

Es posible que vea el siguiente error cuando utilice la aceleración de GPU. Ejecute el siguiente comando para habilitar el acceso a la GPU para el usuario de Greengrass.

```
sudo usermod -a -G video ggc_user
```

No existe tal archivo o directorio

Los siguientes errores indican que el componente de tiempo de ejecución no pudo configurar el entorno virtual correctamente:

- *MLRootPath*`/greengrass_ml_dlr_conda/bin/conda: No such file or directory`
- *MLRootPath*`/greengrass_ml_dlr_venv/bin/activate: No such file or directory`
- *MLRootPath*`/greengrass_ml_tflite_conda/bin/conda: No such file or directory`
- *MLRootPath*`/greengrass_ml_tflite_venv/bin/activate: No such file or directory`

Compruebe los registros para asegurarse de que todas las dependencias del tiempo de ejecución se instalaron correctamente. Para obtener más información acerca de las bibliotecas que instale la cadena de instalación, consulte los temas siguientes:

- Tiempo de ejecución de DLR
- TensorFlow Tiempo de ejecución Lite

De forma predeterminada *MLRootPath* está configurada en*/greengrass/v2*`/work/`*component-name*`/greengrass_ml`. Para cambiar esta ubicación, incluya el componente de tiempo de ejecución Tiempo de ejecución de DLR o TensorFlow Tiempo de ejecución Lite directamente en la implementación y especifique un valor modificado para el parámetro `MLRootPath` en una actualización de combinación de configuraciones. Para obtener más información sobre la configuración de componentes, consulte Actualización de las configuraciones de los componentes.

> **Note**
>
> Para el componente DLR versión 1.3.x, se establece el parámetro `MLRootPath` en la configuración del componente de inferencia y el valor predeterminado es `$HOME/greengrass_ml`.

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

Es posible que aparezcan los siguientes errores cuando ejecute la inferencia de machine learning en Raspberry Pi con el sistema operativo Bullseye de Raspberry Pi.

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

Este error se produce porque Raspberry Pi OS Bullseye incluye una versión anterior a la NumPy que requiere OpenCV. Para solucionar este problema, ejecuta el siguiente comando para actualizar NumPy a la versión más reciente.

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

Es posible que aparezca el siguiente error cuando ejecute la inferencia de machine learning en una Raspberry Pi con el sistema operativo Bullseye de Raspberry Pi.

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
 ensure that the camera has been enabled.
```

Este error se produce porque el sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que no es compatible con los componentes de ML. Para solucionar este problema, habilite la pila de cámara antigua.

Cómo activar la pila de cámara antigua

1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

   ```
   sudo raspi-config
   ```

2. Seleccione Opciones de interfaz.
3. Seleccione Cámara antigua para activar la pila de cámara antigua.
4. Reinicie el Raspberry Pi.

Errores de memoria

Los siguientes errores suelen producirse cuando el dispositivo no tiene suficiente memoria y se interrumpe el proceso de los componentes.

- `stderr. Killed.`
- `exitCode=137`

Recomendamos un mínimo de 500 MB de memoria para implementar un componente de inferencia de machine learning público.

Errores de espacio en el disco

El error `no space left on device` suele ocurrir cuando un dispositivo no tiene suficiente espacio de almacenamiento. Asegúrese de que haya suficiente espacio disponible en el disco en el dispositivo antes de volver a implementar el componente. Recomendamos un mínimo de 500 MB de espacio libre en el disco para implementar un componente de inferencia de machine learning público.

Errores de tiempo de espera

Los componentes públicos de machine learning descargan archivos de modelos de machine learning de gran tamaño que superan los 200 MB. Si se agota el tiempo de espera de la descarga durante la implementación, compruebe la velocidad de la conexión a Internet y vuelva a intentar la implementación.

Administración de los dispositivos principales de Greengrass con AWS Systems Manager

> **Note**
>
> AWS IoT Greengrass actualmente no admite esta característica en los dispositivos principales de Windows.

Systems Manager es un servicio de AWS que puede utilizar para ver y controlar su infraestructura de AWS, que incluye instancias de Amazon EC2, servidores y máquinas virtuales (VM) en las instalaciones y dispositivos de periferia. Systems Manager le permite ver los datos operativos, automatizar las tareas operativas y mantener la seguridad y la conformidad. Cuando se registra una máquina en Systems Manager, se denomina nodo administrado. Para obtener más información, consulte ¿Qué es AWS Systems Manager? en la Guía del usuario de AWS Systems Manager.

El agente de AWS Systems Manager (agente de Systems Manager) es un software que puede instalar en los dispositivos para permitir que Systems Manager los actualice, administre y configure. Para instalar el agente de Systems Manager en los dispositivos principales de Greengrass, implemente el componente agente de Systems Manager. Cuando implemente el agente de Systems Manager por primera vez, este registra el dispositivo principal como un nodo administrado por Systems Manager. El agente de Systems Manager se ejecuta en el dispositivo para permitir la comunicación con el servicio de Systems Manager en la Nube de AWS. Para obtener más información acerca de cómo instalar y configurar el componente agente de Systems Manager, consulte Instalar el AWS Systems Manager agente.

Las herramientas y características de Systems Manager se denominan capacidades. Los dispositivos principales de Greengrass admiten todas las capacidades de Systems Manager. Para obtener más información sobre estas capacidades y sobre cómo usar Systems Manager para administrar los dispositivos principales, consulte Capacidades de Systems Manager en la Guía del usuario de AWS Systems Manager.

AWS Systems Manager ofrece un nivel de instancias estándar y un nivel de instancias avanzadas para los nodos que administra Systems Manager. Si es la primera vez que utiliza Systems Manager, comience con el nivel de instancias estándar. En el nivel de instancias estándar puede registrar hasta 1000 nodos administrados por Región de AWS en su Cuenta de AWS. Si tiene que registrar más

de 1000 nodos administrados en una única cuenta y región, o si tiene que utilizar la capacidad de Session Manager, utilice el nivel de instancias avanzadas. Para obtener más información, consulte Cómo configurar niveles de instancia en la Guía del usuario de AWS Systems Manager.

Temas

- Instalar el AWS Systems Manager agente
- Desinstalar el AWS Systems Manager agente

Instalar el AWS Systems Manager agente

El AWS Systems Manager agente (Systems Manager Agent) es un software de Amazon que se instala para permitir que Systems Manager actualice, gestione y configure los dispositivos principales de Greengrass, las instancias de Amazon EC2 y otros recursos. El agente procesa y ejecuta las solicitudes del servicio de Systems Manager en la Nube de AWS. Luego, el agente envía la información de estado y tiempo de ejecución al servicio de Systems Manager. Para obtener más información, consulte Información del agente de Systems Manager en la Guía del usuario de AWS Systems Manager .

AWS proporciona el agente de Systems Manager como un componente de Greengrass que puede implementar en sus dispositivos principales de Greengrass para administrarlos con Systems Manager. El componente agente de Systems Manager instala el software del agente de Systems Manager y registra el dispositivo principal como un nodo administrado en Systems Manager. Siga los pasos de esta página para completar los requisitos previos e implementar el componente agente de Systems Manager en un dispositivo principal o en un grupo de dispositivos principales.

Temas

- Paso 1: completar los pasos generales de configuración de Systems Manager
- Paso 2: Crear un rol de servicio de IAM para Systems Manager
- Paso 3: Agregar permisos al rol de intercambio de token
- Paso 4: Implementar el componente agente de Systems Manager
- Paso 5: Verificar el registro del dispositivo principal en Systems Manager

Paso 1: completar los pasos generales de configuración de Systems Manager

Si aún no lo ha hecho, complete los pasos generales de configuración de. AWS Systems Manager Para obtener más información, consulte Completar los pasos de configuración general de Systems Manager en la Guía del usuario de AWS Systems Manager .

Paso 2: Crear un rol de servicio de IAM para Systems Manager

El agente de Systems Manager utiliza un rol de servicio AWS Identity and Access Management (IAM) para comunicarse con AWS Systems Managerél. Systems Manager asume este rol para habilitar las capacidades de Systems Manager en cada dispositivo principal. El componente agente de Systems Manager también utiliza este rol para registrar el dispositivo principal como un nodo administrado por Systems Manager al implementar el componente. Si aún no lo ha hecho, cree un rol de servicio de Systems Manager para que lo utilice el componente agente de Systems Manager. Para obtener más información, consulte Creación de un rol de servicio de IAM para dispositivos de periferia en la Guía del usuario de AWS Systems Manager .

Paso 3: Agregar permisos al rol de intercambio de token

Los dispositivos principales de Greengrass utilizan una función de servicio de IAM, denominada función de intercambio de fichas, para interactuar con los servicios. AWS Cada dispositivo principal tiene una función de intercambio de fichas que se crea al instalar el software AWS IoT Greengrass Core. Muchos componentes de Greengrass, como el Agente de Systems Manager, requieren permisos adicionales para este rol. El componente agente de Systems Manager requiere los siguientes permisos, que incluyen el permiso para usar el rol que creó en Paso 2: Crear un rol de servicio de IAM para Systems Manager.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
```

```
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Si aún no lo ha hecho, agregue estos permisos al rol de intercambio de token del dispositivo principal para permitir que funcione el Agente de Systems Manager. Puede agregar una nueva política al rol de intercambio de token para conceder este permiso.

Cómo agregar permisos al rol de intercambio de token (consola)

1. En el menú de navegación de la consola de IAM, elija Roles.
2. Elija la función de IAM que configuró como función de intercambio de fichas al instalar el software AWS IoT Greengrass Core. Si no especificó un nombre para la función de intercambio de fichas al instalar el software AWS IoT Greengrass Core, se creó una función denominada`GreengrassV2TokenExchangeRole`.
3. En la pestaña Permisos, elija Agregar permisos y, a continuación, Asociar políticas.
4. Elija Crear política. Se abre la página Crear política en una nueva pestaña del navegador.
5. En la página Create policy (Crear política), haga lo siguiente:

   a. Elija JSON para abrir el editor JSON.

   b. Pegue la siguiente política de en el editor JSON. *`SSMServiceRole`*Sustitúyalo por el nombre del rol de servicio en el que lo creastePaso 2: Crear un rol de servicio de IAM para Systems Manager.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Action": [
              "iam:PassRole"
            ],
            "Effect": "Allow",
      ```

```
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

c. Elija Siguiente: Etiquetas.

d. Elija Siguiente: Revisar.

e. Introduzca un Nombre para la política, como **`GreengrassSSMAgentComponentPolicy`**.

f. Elija Crear política.

g. Cambie a la pestaña anterior del navegador en la que tenía abierto el rol de intercambio de token.

6. En la página Agregar permisos, pulse el botón de actualización y, a continuación, seleccione la política de agente de Systems Manager en Greengrass que creó en el paso anterior.

7. Seleccione Asociar políticas.

   Los dispositivos principales que utilizan este rol de intercambio de token ahora tienen permiso para interactuar con el servicio Systems Manager.

Cómo agregar permisos al rol de intercambio de token (AWS CLI)

Cómo agregar una política que conceda permiso para usar Systems Manager

1. Cree un archivo llamado `ssm-agent-component-policy.json` y copie el siguiente JSON en el archivo. *`SSMServiceRole`*Sustitúyalo por el nombre del rol de servicio en el que lo creastePaso 2: Crear un rol de servicio de IAM para Systems Manager.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
   ```

```
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

2. Ejecute el siguiente comando para crear la política del documento de política en `ssm-agent-component-policy.json`.

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name GreengrassSSMAgentComponentPolicy \
     --policy-document file://ssm-agent-component-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name GreengrassSSMAgentComponentPolicy ^
     --policy-document file://ssm-agent-component-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name GreengrassSSMAgentComponentPolicy `
     --policy-document file://ssm-agent-component-policy.json
   ```

Copie la política del nombre de recurso de Amazon (ARN) de la política de los metadatos de salida. Utilice este ARN para asociar la política al rol del dispositivo principal en el siguiente paso.

3. Ejecute el siguiente comando para asociar la política al rol de intercambio de token.

   - *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre de la función de intercambio de fichas que especificó al instalar el software AWS IoT Greengrass principal. Si no especificó un nombre para la función de intercambio de fichas al instalar el software AWS IoT Greengrass Core, se creó una función denominada`GreengrassV2TokenExchangeRole`.
   - Sustituya el ARN de la política por el ARN del paso anterior.

   Linux or Unix

   ```
   aws iam attach-role-policy \
     --role-name GreengrassV2TokenExchangeRole \
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam attach-role-policy ^
     --role-name GreengrassV2TokenExchangeRole ^
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   PowerShell

   ```
   aws iam attach-role-policy `
     --role-name GreengrassV2TokenExchangeRole `
     --policy-arn
    arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
   ```

   Si todo es correcto, el comando no tiene salida. Los dispositivos principales que utilizan este rol de intercambio de token ahora tienen permiso para interactuar con el servicio Systems Manager.

Paso 4: Implementar el componente agente de Systems Manager

Complete los siguientes pasos para implementar y configurar el componente agente de Systems Manager. Puede implementar el componente en un único dispositivo principal o en un grupo de dispositivos principales.

Cómo implementar el componente agente de Systems Manager (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, elija la pestaña Componentes públicos y, luego, elija aws.greengrass.SystemsManagerAgent.
3. En la página aws.greengrass.SystemsManagerAgent, elija Implementar.
4. En Agregar a la implementación, elija una implementación existente para revisarla o cree una nueva y, a continuación, elija Siguiente.
5. Si opta por crear una nueva implementación, elija el dispositivo principal o el grupo de objetos de destino para la implementación. En la página Especificar el destino, en Destino de la implementación, elija un dispositivo principal o un grupo de objetos y, a continuación, elija Siguiente.
6. En la página Seleccionar componentes, compruebe que el componente aws.greengrass.SystemsManagerAgent esté seleccionado y elija Siguiente.
7. En la página Configurar componentes, seleccione aws.greengrass.SystemsManagerAgent y haga lo siguiente:

   a. Seleccione Configurar componente.

   b. En el cuadro Configurar aws.greengrass.SystemsManagerAgent, en Actualización de configuración, en Configuración para combinar, ingrese la siguiente actualización de configuración. *SSMServiceRole*Sustitúyalo por el nombre del rol de servicio en el que lo creastePaso 2: Crear un rol de servicio de IAM para Systems Manager.

      ```
      {
        "SSMRegistrationRole": "SSMServiceRole",
        "SSMOverrideExistingRegistration": false
      }
      ```

> **Note**
>
> Si el dispositivo principal ya ejecuta el Agente de Systems Manager registrado con una activación híbrida, cambie `SSMOverrideExistingRegistration` a `true`. Este parámetro especifica si el componente agente de Systems Manager registra el dispositivo principal cuando el Agente de Systems Manager ya se está ejecutando en el dispositivo con una activación híbrida.
> También puede especificar etiquetas (`SSMResourceTags`) para agregarlas al nodo administrado por Systems Manager que el componente agente de Systems Manager crea para el dispositivo principal. Para obtener más información, consulte Configuración del componente agente de Systems Manager.

c. Elija Confirmar para cerrar el cuadro y, a continuación, elija Siguiente.

8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Revisar, elija Implementar.

   La implementación puede tardar hasta un minuto para completarse.

Cómo implementar el componente agente de Systems Manager (AWS CLI)

Para implementar el componente agente de Systems Manager, cree un documento de implementación que incluya `aws.greengrass.SystemsManagerAgent` en el objeto `components` y especifique la actualización de configuración del componente. Siga las instrucciones en Crear implementaciones para crear una implementación nueva o revisar una implementación existente.

El siguiente ejemplo de documento de implementación parcial especifica el uso de un rol de servicio denominado `SSMServiceRole`. *`SSMServiceRole`*Sustitúyalo por el nombre del rol de servicio en el que lo creastePaso 2: Crear un rol de servicio de IAM para Systems Manager.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
```

```
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

> **Note**
>
> Si el dispositivo principal ya ejecuta el Agente de Systems Manager registrado con una activación híbrida, cambie `SSMOverrideExistingRegistration` a `true`. Este parámetro especifica si el componente agente de Systems Manager registra el dispositivo principal cuando el Agente de Systems Manager ya se está ejecutando en el dispositivo con una activación híbrida.
>
> También puede especificar etiquetas (`SSMResourceTags`) para agregarlas al nodo administrado por Systems Manager que el componente agente de Systems Manager crea para el dispositivo principal. Para obtener más información, consulte Configuración del componente agente de Systems Manager.

La implementación puede tardar varios minutos en completarse. Puede utilizar el AWS IoT Greengrass servicio para comprobar el estado de la implementación y comprobar los registros del software AWS IoT Greengrass principal y los registros de los componentes del Agente de Systems Manager para comprobar que el Agente de Systems Manager se ejecuta correctamente. Para obtener más información, consulte los siguientes temas:

- Comprobación del estado de la implementación
- Supervisión de los registros de AWS IoT Greengrass
- Visualización de los registros del agente de Systems Manager en la Guía del usuario de AWS Systems Manager

Si la implementación falla o el Agente de Systems Manager no se ejecuta, puede solucionar los problemas de la implementación en cada dispositivo principal. Para obtener más información, consulte los siguientes temas:

- Solución de problemas AWS IoT Greengrass V2

- Solución de problemas del Agente de Systems Manager en la Guía del usuario de AWS Systems Manager

Paso 5: Verificar el registro del dispositivo principal en Systems Manager

Cuando se ejecuta el componente agente de Systems Manager, registra el dispositivo principal como nodo administrado en Systems Manager. Puede usar la AWS IoT Greengrass consola, la consola de Systems Manager y la API de Systems Manager para comprobar que un dispositivo principal esté registrado como nodo gestionado. Los nodos administrados también se denominan instancias en partes de la consola de AWS y la API.

Para verificar el registro del dispositivo principal (AWS IoT Greengrass consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Dispositivos principales.
2. Elija el dispositivo principal que desee verificar.
3. En la página de detalles del dispositivo principal, busque la propiedad de la Instancia de AWS Systems Manager . Si esta propiedad está presente y muestra un enlace a la consola de Systems Manager, el dispositivo principal está registrado como nodo administrado.

   También puede encontrar la propiedad estado de ping AWS Systems Manager para comprobar el estado del Agente de Systems Manager en el dispositivo principal. Cuando el estado es En línea, puede administrar el dispositivo principal con Systems Manager.

Cómo verificar el registro del dispositivo principal (consola de Systems Manager)

1. En el menú de navegación de la consola de Systems Manager, seleccione Administrador de flotas.
2. En Nodos administrados, haga lo siguiente:
   a. Agregue un filtro en el que Tipo de origen sea AWS::IoT::Thing.
   b. Agregue un filtro en el que ID de origen sea el nombre del dispositivo principal que se va a verificar.
3. Busque el dispositivo principal en la tabla de Nodos administrados. Si el dispositivo principal está en la tabla, está registrado como nodo administrado.

También puede encontrar la propiedad estado de ping del agente de Systems Manager para comprobar el estado del Agente de Systems Manager en el dispositivo principal. Cuando el estado es En línea, puede administrar el dispositivo principal con Systems Manager.

Cómo verificar el registro del dispositivo principal (AWS CLI)

- Utilice la DescribeInstanceInformationoperación para obtener la lista de nodos gestionados que coinciden con el filtro que especifique. Ejecute el siguiente comando para verificar si un dispositivo principal está registrado como nodo administrado. *MyGreengrassCore*Sustitúyalo por el nombre del dispositivo principal para verificarlo.

  ```
  aws ssm describe-instance-information --filter
   Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
  ```

  La respuesta contiene la lista de nodos administrados que coinciden con el filtro. Si la lista contiene un nodo administrado, el dispositivo principal se registra como nodo administrado. También puede encontrar más información sobre el nodo administrado del dispositivo principal en la respuesta. Cuando la propiedad `PingStatus` está `Online`, puede administrar el dispositivo principal con Systems Manager.

Tras comprobar que un dispositivo principal está registrado como nodo administrado en Systems Manager, puede utilizar la consola y la API de Systems Manager para administrar ese dispositivo principal. Para obtener más información sobre estas capacidades de Systems Manager que puede usar para administrar los dispositivos principales de Greengrass, consulte Capacidades de Systems Manager en la Guía del usuario de AWS Systems Manager .

Desinstalar el AWS Systems Manager agente

Si ya no desea administrar un dispositivo principal de Greengrass AWS Systems Manager, puede anular el registro del dispositivo principal en Systems Manager y desinstalar el agente (AWS Systems Manager Systems Manager Agent) del dispositivo.

Puede volver a registrar un dispositivo principal en cualquier momento. Para ello, vuelva a implementar el componente Agente de Systems Manager, que registra el dispositivo principal en Systems Manager cuando se instala. Systems Manager almacena el historial de comandos de un dispositivo principal cancelado durante 30 días.

Temas

- Paso 1: Anular el registro del dispositivo principal de Systems Manager
- Paso 2: Desinstalar el componente agente de Systems Manager
- Paso 3: Desinstalar el software agente de Systems Manager

Paso 1: Anular el registro del dispositivo principal de Systems Manager

Puede utilizar la consola o la API de Systems Manager para anular el registro del dispositivo principal. Para obtener más información, consulte Grupos de nodos administrados en la Guía del usuario de AWS Systems Manager .

Paso 2: Desinstalar el componente agente de Systems Manager

Tras anular el registro del dispositivo principal, desinstale el componente agente de Systems Manager del dispositivo. Para eliminar un componente de un dispositivo principal de Greengrass, revise la implementación en la que se instaló el componente y elimínelo de la implementación. El software AWS IoT Greengrass principal desinstala un componente cuando ninguna de las implementaciones de un dispositivo principal especifica ese componente. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Cómo desinstalar el componente agente de Systems Manager (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Dispositivos principales.
2. Elija el dispositivo principal en el que desee desinstalar el componente agente de Systems Manager.
3. En la página de detalles del dispositivo principal, elija la pestaña Implementaciones.
4. Elija la implementación que implementa el componente agente de Systems Manager en el dispositivo principal.
5. En la página de detalles de la implementación, elija Revisar.
6. En el cuadro Revisar implementación, elija Revisar implementación.
7. En el Paso 1: Especificar el objetivo, seleccione Siguiente.
8. En el Paso 2: Seleccionar los componentes, borre la selección del componente aws.greengrass.SystemsManagerAgent y, a continuación, elija Siguiente.
9. En el Paso 3: Configurar los componentes, seleccione Siguiente.
10. En el Paso 4: Configurar los ajustes avanzados, seleccione Siguiente.

11. En el Paso 5: Revisar, elija Implementar.

Cómo desinstalar el componente agente de Systems Manager (CLI)

Para desinstalar el componente agente de Systems Manager, revise la implementación que lo implementa y elimínelo de la implementación. Para obtener más información, consulte Revisión de las implementaciones.

La implementación puede tardar varios minutos en completarse. Puede utilizar el AWS IoT Greengrass servicio para comprobar el estado de la implementación. Para obtener más información, consulte Comprobación del estado de la implementación.

Paso 3: Desinstalar el software agente de Systems Manager

El software agente de Systems Manager continúa ejecutándose en el dispositivo principal después de eliminar el componente agente de Systems Manager. Para eliminar el software agente de Systems Manager, puede ejecutar comandos en el dispositivo principal. Para obtener más información, consulte Desinstalación manual del agente de Systems Manager en instancias de Linux en la Guía del usuario de AWS Systems Manager .

Seguridad en AWS IoT Greengrass

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre usted AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la Nube de AWS. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener más información sobre los programas de cumplimiento aplicables AWS IoT Greengrass, consulte AWS Servicios incluidos en el ámbito de aplicación por programa de conformidad y AWS servicios incluidos .
- Seguridad en la nube: su responsabilidad viene determinada por el servicio de AWS que utilice. También es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Cuando lo usa AWS IoT Greengrass, también es responsable de proteger sus dispositivos, la conexión de red local y las claves privadas.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza AWS IoT Greengrass. Los siguientes temas muestran cómo configurarlo AWS IoT Greengrass para cumplir sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros AWS servicios que le ayudan a supervisar y proteger sus AWS IoT Greengrass recursos.

Temas

- Protección de datos en AWS IoT Greengrass
- Autenticación y autorización de dispositivos para AWS IoT Greengrass
- Administración de identidad y acceso para AWS IoT Greengrass
- Cómo permitir el tráfico del dispositivo a través de un proxy o firewall
- Validación de conformidad en AWS IoT Greengrass
- Puntos de conexión de FIPS
- Resiliencia en AWS IoT Greengrass

- Seguridad de la infraestructura en AWS IoT Greengrass
- Análisis de configuración y vulnerabilidad en AWS IoT Greengrass
- Integridad del código en AWS IoT Greengrass V2
- AWS IoT Greengrass y puntos finales de VPC de interfaz ()AWS PrivateLink
- Mejores prácticas de seguridad para AWS IoT Greengrass

Protección de datos en AWS IoT Greengrass

El modelo de responsabilidad AWS compartida modelo se aplica a la protección de datos en AWS IoT Greengrass. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global que ejecuta todos los Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el Modelo de responsabilidad compartida de AWS y GDPR en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Se utiliza SSL/TLS para comunicarse con AWS los recursos. Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte Cómo trabajar con CloudTrail senderos en la Guía del AWS CloudTrail usuario.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger la información confidencial almacenada en Amazon S3.

- Si necesita módulos criptográficos validados por FIPS 140-3 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-3.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabaja con AWS IoT Greengrass o Servicios de AWS utiliza la consola, la API o AWS CLI AWS SDKs Cualquier dato que introduzca en etiquetas o campos de formato libre utilizados para los nombres se pueden emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Para obtener más información sobre cómo proteger la información confidencial en AWS IoT Greengrass, consultethe section called “No registre información confidencial”.

Para obtener más información sobre la protección de datos, consulte la entrada de blog relativa al modelo de responsabilidad compartida de AWS y GDPR en el blog de seguridad de AWS .

Temas

- Cifrado de datos
- Integración de la seguridad de hardware

Cifrado de datos

AWS IoT Greengrass utiliza el cifrado para proteger los datos mientras están en tránsito (a través de Internet o una red local) y en reposo (almacenados en la Nube de AWS).

Los dispositivos de un AWS IoT Greengrass entorno suelen recopilar datos que se envían a AWS los servicios para su posterior procesamiento. Para obtener más información sobre el cifrado de datos en otros AWS servicios, consulte la documentación de seguridad de ese servicio.

Temas

- Cifrado en tránsito
- Cifrado en reposo
- Administración de claves en el dispositivo del núcleo de Greengrass

Cifrado en tránsito

AWS IoT Greengrass tiene dos modos de comunicación donde los datos están en tránsito:

- the section called “Datos en tránsito a través de Internet”. La comunicación entre un núcleo de Greengrass y AWS IoT Greengrass a través de Internet está cifrada.
- the section called “Datos del dispositivo central”. La comunicación entre componentes en el dispositivo del núcleo de Greengrass no está cifrada.

Datos en tránsito a través de Internet

AWS IoT Greengrass utiliza seguridad de la capa de transporte (TLS) para cifrar toda la comunicación a través de Internet. Todos los datos enviados a la Nube de AWS se envían a través de una conexión TLS utilizando protocolos MQTT o HTTPS, por lo que es seguro de forma predeterminada. AWS IoT Greengrass utiliza el modelo de seguridad de transporte de AWS IoT. Para obtener más información, consulte Seguridad de transporte en la Guía del desarrollador de AWS IoT Core.

Datos del dispositivo central

AWS IoT Greengrass no cifra los datos intercambiados localmente en el dispositivo del núcleo de Greengrass porque los datos no salen del dispositivo. Esto incluye la comunicación entre componentes definidos por el usuario, el SDK del dispositivo de AWS IoT y componentes públicos, como el administrador de flujos.

Cifrado en reposo

AWS IoT Greengrass almacena los datos:

- the section called “Datos en reposo en la Nube de AWS”. Estos datos están cifrados.
- the section called “Datos en reposo en el núcleo de Greengrass”. Estos datos no están cifrados (excepto las copias locales de sus secretos).

Datos en reposo en la Nube de AWS

AWS IoT Greengrass cifra los datos de los clientes almacenados en la Nube de AWS. Estos datos están protegidos mediante claves de AWS KMS administradas por AWS IoT Greengrass.

Datos en reposo en el núcleo de Greengrass

AWS IoT Greengrass se basa en permisos de archivos Unix y cifrado de disco completo (si está habilitado) para proteger los datos en reposo en el núcleo. Tiene la responsabilidad de proteger el sistema de archivos y el dispositivo.

Sin embargo, AWS IoT Greengrass cifra las copias locales de sus secretos recuperados de AWS Secrets Manager. Para obtener más información, consulte el componente administrador de secretos.

Administración de claves en el dispositivo del núcleo de Greengrass

Es responsabilidad del cliente garantizar el almacenamiento seguro de claves criptográficas (públicas y privadas) en el dispositivo del núcleo de Greengrass. AWS IoT Greengrass utiliza claves públicas y privadas para la siguiente situación:

- La clave de cliente de IoT se utiliza con el certificado IoT para autenticar el protocolo de enlace Transport Layer Security (TLS) cuando un núcleo de Greengrass se conecta a AWS IoT Core. Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

  > **Note**
  >
  > La clave y el certificado también se conocen como clave privada del núcleo y el certificado de dispositivo del núcleo.

Un dispositivo principal de Greengrass admite el almacenamiento de claves privadas mediante permisos del sistema de archivos o en un módulo de seguridad de hardware. Si utiliza claves privadas basadas en el sistema de archivos, es responsable de su almacenamiento seguro en el dispositivo del núcleo.

Integración de la seguridad de hardware

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Puede configurar el software AWS IoT Greengrass Core para que utilice un módulo de seguridad de hardware (HSM) a través de la interfaz PKCS #11. Esta característica le permite almacenar de forma segura los certificados y claves privadas del dispositivo para que no queden expuestos ni duplicados en el software. Puede almacenar la clave privada y el certificado en un módulo de hardware, como un HSM o un módulo de plataforma segura (TPM).

El software AWS IoT Greengrass Core utiliza una clave privada y un certificado X.509 para autenticar las conexiones a los servicios y. AWS IoT AWS IoT Greengrass El componente administrador de secretos utiliza esta clave privada para cifrar y descifrar de forma segura los secretos que se implementan en un dispositivo principal de Greengrass. Al configurar un dispositivo principal para usar un HSM, estos componentes utilizan la clave privada y el certificado que se almacenan en el HSM.

El componente agente MQTT de Moquette también almacena una clave privada para su certificado de servidor MQTT local. Este componente almacena la clave privada en el sistema de archivos del dispositivo, en la carpeta de trabajo del componente. Actualmente, AWS IoT Greengrass no admite el almacenamiento de esta clave privada o certificado en un HSM.

> **Tip**
>
> Busque los dispositivos que admiten esta característica en el Catálogo de dispositivos de socios de AWS.

Temas

- Requisitos
- Prácticas recomendadas de seguridad de hardware
- AWS IoT Greengrass Instale el software principal con seguridad de hardware
- Configuración de la seguridad del hardware en un dispositivo principal existente
- Uso del hardware sin compatibilidad con PKCS#11
- Véase también

Requisitos

Debe cumplir con los siguientes requisitos para usar un HSM en un dispositivo principal de Greengrass:

- Núcleo de Greengrass versión 2.5.3 o posterior instalado en el dispositivo principal. Puede elegir una versión compatible al instalar el software AWS IoT Greengrass Core en un dispositivo principal.
- El componente del proveedor PKCS#11 instalado en el dispositivo principal. Puede descargar e instalar este componente al instalar el software AWS IoT Greengrass principal en un dispositivo principal.
- Un módulo de seguridad de hardware que admite el esquema de firmas PKCS#1 versión 1.5 y claves RSA con un tamaño de clave RSA-2048 (o mayor) o claves ECC.

  > **Note**
  >
  > Para utilizar un módulo de seguridad de hardware con claves ECC, debe utilizar la versión del núcleo de Greengrass 2.5.6 o posterior.
  > Para usar un módulo de seguridad de hardware y el administrador de secretos, debe usar un módulo de seguridad de hardware con claves RSA.

- Una biblioteca de proveedores de PKCS #11 que el software AWS IoT Greengrass principal puede cargar en tiempo de ejecución (mediante libdl) para invocar las funciones de PKCS #11. La biblioteca de proveedores PKCS#11 debe implementar las siguientes operaciones de la API de PKCS#11:
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`

  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- El módulo de hardware debe resolverlo la etiqueta de ranura, tal y como se define en la especificación de PKCS#11.
- Debe almacenar la clave privada y el certificado en el HSM, en la misma ranura, y deben usar la misma etiqueta de objeto e ID de objeto, si el HSM admite el objeto. IDs
- El certificado y la clave privada debe poder resolverla con etiquetas de objeto.
- La clave privada debe tener los siguientes permisos:
  - `sign`
  - `decrypt`
- (Opcional) Para usar el componente administrador de secretos, debe usar la versión 2.1.0 o posterior y la clave privada debe tener los siguientes permisos:
  - `unwrap`
  - `wrap`

Prácticas recomendadas de seguridad de hardware

Tenga en cuenta las siguientes prácticas recomendadas al configurar la seguridad del hardware en los dispositivos principales de Greengrass.

- Genere claves privadas directamente en el HSM utilizando el generador de números aleatorios de hardware interno. Este enfoque es más seguro que importar una clave privada que se genere en otro lugar, ya que la clave privada permanece dentro del HSM.
- Configure las claves privadas para que sean inmutables y prohíba la exportación.
- Utilice la herramienta de aprovisionamiento que el proveedor de hardware de HSM recomiende para generar una solicitud de firma de certificado (CSR) con la clave privada protegida por hardware y, a continuación, utilice la consola o la API para generar un certificado de cliente. AWS IoT

> **Note**
>
> La práctica recomendada de seguridad para la rotación de claves no se aplica cuando las claves privadas se generan en un HSM.

AWS IoT Greengrass Instale el software principal con seguridad de hardware

Al instalar el software AWS IoT Greengrass Core, puede configurarlo para que utilice una clave privada que genere en un HSM. Este enfoque sigue las prácticas recomendadas de seguridad para generar la clave privada en el HSM, de modo que la clave privada permanezca dentro del HSM.

Para instalar el software AWS IoT Greengrass Core con seguridad de hardware, haga lo siguiente:

1. Genere una clave privada en los HSM.
2. Cree una solicitud de firma de certificado (CSR) de la clave privada.
3. Cree un certificado de cliente a partir de la CSR. Puede crear un certificado firmado por AWS IoT o por otra autoridad de certificación (CA) raíz. Para obtener más información sobre cómo usar otra CA raíz, consulte Crear sus propios certificados de cliente en la Guía para desarrolladores de AWS IoT Core .
4. Descargue el AWS IoT certificado e impórtelo al HSM.
5. Instale el software AWS IoT Greengrass principal desde un archivo de configuración que especifique el uso del componente proveedor PKCS #11 y la clave privada y el certificado en el HSM.

Puede elegir una de las siguientes opciones de instalación para instalar el software AWS IoT Greengrass principal con seguridad de hardware:

- Instalación manual

  Elija esta opción para crear manualmente los AWS recursos necesarios y configurar la seguridad del hardware. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.
- Instalación con aprovisionamiento personalizado

  Elija esta opción para desarrollar una aplicación Java personalizada que cree automáticamente los AWS recursos necesarios y configure la seguridad del hardware. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento de recursos personalizado.

Actualmente, AWS IoT Greengrass no admite la instalación del software AWS IoT Greengrass principal con seguridad de hardware cuando se instala con el aprovisionamiento automático de recursos o el aprovisionamiento de AWS IoT flotas.

Configuración de la seguridad del hardware en un dispositivo principal existente

Puede importar la clave privada y el certificado de un dispositivo principal a un HSM para configurar la seguridad del hardware.

> **Consideraciones**
>
> - Debe tener acceso raíz al sistema de archivos del dispositivo principal.
> - En este procedimiento, se apaga el software AWS IoT Greengrass principal para que el dispositivo principal quede desconectado y no esté disponible mientras se configura la seguridad del hardware.

Para configurar la seguridad del hardware en un dispositivo principal existente, haga lo siguiente:

1. Inicialice el HSM.
2. Implemente el componente del proveedor PKCS#11 en el dispositivo principal.
3. Detenga el software AWS IoT Greengrass principal.
4. Importe la clave privada y el certificado del dispositivo principal al HSM.
5. Actualice el archivo de configuración del software AWS IoT Greengrass principal para usar la clave privada y el certificado del HSM.

6. Inicie el software AWS IoT Greengrass principal.

Paso 1: Inicializar el módulo de seguridad de hardware

Complete el siguiente paso para inicializar el HSM en su dispositivo principal.

Cómo inicializar el módulo de seguridad de hardware

- Inicialice un token PKCS#11 en el HSM y guarde el ID de la ranura y el PIN de usuario correspondientes al token. Consulte la documentación de su HSM para obtener información sobre cómo inicializar un token. El ID de ranura y el PIN de usuario se utilizan más adelante al implementar y configurar el componente del proveedor PKCS#11.

Paso 2: Implementar el componente del proveedor PKCS#11

Complete los siguientes pasos para implementar y configurar el componente del proveedor PKCS#11. Puede implementar el componente en uno o más dispositivos principales.

Cómo implementar el componente del proveedor PKCS#11 (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Componentes.
2. En la página Componentes, elija la pestaña Componentes públicos y, luego, elija aws.greengrass.crypto.Pkcs11Provider.
3. En la página aws.greengrass.crypto.Pkcs11Provider, elija Implementar.
4. En Agregar a la implementación, elija una implementación existente para revisarla o cree una nueva y, a continuación, elija Siguiente.
5. Si opta por crear una nueva implementación, elija el dispositivo principal o el grupo de objetos de destino para la implementación. En la página Especificar el destino, en Destino de la implementación, elija un dispositivo principal o un grupo de objetos y, a continuación, elija Siguiente.
6. En la página Seleccionar componentes, en Componentes públicos, seleccione aws.greengrass.crypto.Pkcs11Provider y, a continuación, elija Siguiente.
7. En la página Configurar componentes, seleccione aws.greengrass.crypto.Pkcs11Provider y haga lo siguiente:

   a. Seleccione Configurar componente.

b. En el cuadro Configurar aws.greengrass.crypto.Pkcs11Provider, en Actualización de configuración, en Configuración para combinar, ingrese la siguiente actualización de configuración. Actualice los siguientes parámetros de configuración con los valores de los dispositivos principales de destino. Especifique el ID de ranura y el PIN de usuario en los que inicializó anteriormente el token PKCS#11. Más adelante, importará la clave privada y el certificado a esta ranura del HSM.

`name`

Un nombre para la configuración de PKCS#11.

`library`

La ruta absoluta del archivo a la biblioteca de la implementación del PKCS #11 que el software AWS IoT Greengrass Core puede cargar con libdl.

`slot`

El ID de la ranura que contiene la clave privada y el certificado del dispositivo. Este valor es diferente del índice o la etiqueta de la ranura.

`userPin`

El PIN del usuario que se utiliza para acceder a la ranura.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

c. Elija Confirmar para cerrar el cuadro y, a continuación, elija Siguiente.

8. En la página Configurar ajustes avanzados, mantenga los ajustes de configuración predeterminados y seleccione Siguiente.
9. En la página Revisar, elija Implementar.

La implementación puede tardar hasta un minuto para completarse.

Cómo implementar el componente del proveedor PKCS#11 (AWS CLI)

Para implementar el componente del proveedor PKCS#11, cree un documento de implementación que incluya `aws.greengrass.crypto.Pkcs11Provider` en el objeto `components` y especifique la actualización de configuración del componente. Siga las instrucciones en Crear implementaciones para crear una implementación nueva o revisar una implementación existente.

El siguiente ejemplo de documento de implementación parcial especifica la implementación y la configuración del componente del proveedor PKCS#11. Actualice los siguientes parámetros de configuración con los valores de los dispositivos principales de destino. Guarde el ID de ranura y el PIN de usuario para usarlos más adelante cuando importe la clave privada y el certificado al HSM.

`name`

Un nombre para la configuración de PKCS#11.

`library`

La ruta absoluta del archivo a la biblioteca de la implementación de PKCS #11 que el software AWS IoT Greengrass Core puede cargar con libdl.

`slot`

El ID de la ranura que contiene la clave privada y el certificado del dispositivo. Este valor es diferente del índice o la etiqueta de la ranura.

`userPin`

El PIN del usuario que se utiliza para acceder a la ranura.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
```

```
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/
libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
}
```

La implementación puede tardar varios minutos en completarse. Puede usar el AWS IoT Greengrass servicio para comprobar el estado de la implementación. Puede consultar los registros del software AWS IoT Greengrass principal para comprobar que el componente del proveedor PKCS #11 se implementa correctamente. Para obtener más información, consulte los siguientes temas:

- Comprobación del estado de la implementación
- Supervisión de los registros de AWS IoT Greengrass

Si la implementación falla, puede solucionar el problema de la implementación en cada dispositivo principal. Para obtener más información, consulte Solución de problemas AWS IoT Greengrass V2.

Paso 3: Actualizar la configuración en el dispositivo principal

El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica cómo funciona el dispositivo. Este archivo de configuración incluye dónde encontrar la clave privada y el certificado que el dispositivo utiliza para conectarse a la Nube de AWS. Complete los siguientes pasos para importar la clave privada y el certificado del dispositivo principal al HSM y actualice el archivo de configuración para usar el HSM.

Cómo actualizar la configuración del dispositivo principal para utilizar la seguridad del hardware

1. Detenga el software AWS IoT Greengrass principal. Si configuró el software AWS IoT Greengrass Core como un servicio del sistema con systemd, puede ejecutar el siguiente comando para detener el software.

   ```
   sudo systemctl stop greengrass.service
   ```

2. Busque los archivos de clave privada y certificado del dispositivo principal.

   - Si instaló el software AWS IoT Greengrass Core con el aprovisionamiento automático o el aprovisionamiento de flota, la clave privada está en */greengrass/v2*`/privKey.key` y el certificado está en. */greengrass/v2*`/thingCert.crt`

- Si instaló el software AWS IoT Greengrass Core con el aprovisionamiento manual, la clave privada existe de forma */greengrass/v2*/private.pem.key predeterminada y el certificado existe de forma predeterminada. */greengrass/v2*/device.pem.crt

También puede comprobar las propiedades `system.privateKeyPath` y `system.certificateFilePath` en */greengrass/v2*/config/effectiveConfig.yaml para encontrar la ubicación de estos archivos.

3. Importe la clave privada y el certificado al HSM. Consulte la documentación de su HSM para obtener información sobre cómo importar claves privadas y certificados hacia él. Importe la clave privada y el certificado con el ID de ranura y el PIN de usuario en los que inicializó anteriormente el token PKCS#11. Debe usar la misma etiqueta de objeto e ID de objeto para la clave privada y el certificado. Guarde la etiqueta de objeto que especifique al importar cada archivo. Esta etiqueta se utiliza más adelante cuando se actualiza la configuración del software AWS IoT Greengrass principal para utilizar la clave privada y el certificado en el HSM.
4. Actualice la configuración AWS IoT Greengrass principal para usar la clave privada y el certificado en el HSM. Para actualizar la configuración, modifique el archivo de configuración AWS IoT Greengrass principal y ejecute el software AWS IoT Greengrass principal con el archivo de configuración actualizado para aplicar la nueva configuración.

   Haga lo siguiente:

   a. Cree una copia de seguridad del archivo de configuración AWS IoT Greengrass principal. Puede utilizar esta copia de seguridad para restaurar el dispositivo principal si tiene problemas al configurar la seguridad del hardware.

      ```
      sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
      ```

   b. Abra el archivo de configuración de AWS IoT Greengrass Core en un editor de texto. Por ejemplo, puede ejecutar el comando siguiente para usar GNU nano para editar el archivo. Reemplace */greengrass/v2* con la ruta a la carpeta raíz de Greengrass.

      ```
      sudo nano /greengrass/v2/config/effectiveConfig.yaml
      ```

   c. Reemplace el valor de `system.privateKeyPath` por el URI PKCS#11 para la clave privada del HSM. *iotdevicekey*Sustitúyala por la etiqueta del objeto en la que importaste anteriormente la clave privada y el certificado.

```
pkcs11:object=iotdevicekey;type=private
```

d. Reemplace el valor de `system.certificateFilePath` por el URI PKCS#11 del certificado en el HSM. *`iotdevicekey`*Sustitúyala por la etiqueta de objeto en la que importaste anteriormente la clave privada y el certificado.

```
pkcs11:object=iotdevicekey;type=cert
```

Una vez finalizados estos pasos, la `system` propiedad del archivo de configuración AWS IoT Greengrass principal debería tener un aspecto similar al del ejemplo siguiente.

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. Aplique la configuración en el archivo `effectiveConfig.yaml` actualizado. Ejecute `Greengrass.jar` con el parámetro `--init-config` en el que desee aplicar la configuración en `effectiveConfig.yaml`. Reemplace */greengrass/v2* con la ruta a la carpeta raíz de Greengrass.

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. Inicie el software AWS IoT Greengrass Core. Si configuró el software AWS IoT Greengrass Core como un servicio del sistema con systemd, puede ejecutar el siguiente comando para iniciar el software.

```
sudo systemctl start greengrass.service
```

Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

7. Compruebe los registros del software AWS IoT Greengrass principal para comprobar que el software se inicia y se conecta al Nube de AWS. El software AWS IoT Greengrass principal

utiliza la clave privada y el certificado para conectarse a los AWS IoT Greengrass servicios AWS IoT y.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Los siguientes mensajes de registro a nivel de información indican que el software AWS IoT Greengrass principal se ha conectado correctamente a los servicios AWS IoT and AWS IoT Greengrass .

```
2021-12-06T22:47:53.702Z [INFO] (Thread-3)
 com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
 Core. {clientId=MyGreengrassCore5, sessionPresent=false}
```

8. (Opcional) Tras comprobar que el software AWS IoT Greengrass principal funciona con la clave privada y el certificado del HSM, elimine la clave privada y los archivos de certificado del sistema de archivos del dispositivo. Ejecute el siguiente comando y reemplace las rutas de los archivos por las rutas a los archivos de clave privada y certificado.

```
sudo rm /greengrass/v2/privKey.key
sudo rm /greengrass/v2/thingCert.crt
```

Uso del hardware sin compatibilidad con PKCS#11

La biblioteca de PKCS#11 suele ser proporcionada por el proveedor de hardware o es de código abierto. Por ejemplo, con un hardware que cumpla con los estándares (como el TPM1 .2), podría ser posible utilizar el software de código abierto existente. Si el hardware no tiene la correspondiente implementación de la biblioteca PKCS#11, o bien si desea escribir un proveedor de PKCS#11 personalizado, póngase en contacto con su representante de Enterprise Support de Amazon Web Services para plantearle cualquier pregunta relacionada con la integración.

Véase también

- PKCS#11 Cryptographic Token Interface Usage Guide Version 2.4.0
- RFC 7512
- PKCS #1: Cifrado RSA versión 1.5

Autenticación y autorización de dispositivos para AWS IoT Greengrass

Los dispositivos de AWS IoT Greengrass los entornos utilizan certificados X.509 para la autenticación y AWS IoT políticas de autorización. Los certificados y las políticas permiten que los dispositivos se conecten de forma segura entre sí, con AWS IoT Core y AWS IoT Greengrass.

Los certificados X.509 son certificados digitales que utilizan el estándar de infraestructura de clave pública X.509 para asociar una clave pública a una identidad contenida en un certificado. Una entidad de confianza conocida como entidad de certificación (CA) emite los certificados X.509. La CA administra uno o varios certificados especiales llamados certificados de CA, que utiliza para generar certificados X.509. Solo la entidad de certificación tiene acceso a los certificados de entidad de certificación.

AWS IoT las políticas definen el conjunto de operaciones permitidas para AWS IoT los dispositivos. En concreto, permiten y deniegan el acceso y las operaciones del plano de AWS IoT Greengrass datos, como la publicación de mensajes MQTT y la recuperación de imágenes ocultas de los dispositivos. AWS IoT Core

Todos los dispositivos requieren una entrada en el AWS IoT Core registro y un certificado X.509 activado con una política adjunta. AWS IoT Los dispositivos se dividen en dos categorías:

- Dispositivos principales de Greengrass

  Los dispositivos principales de Greengrass utilizan certificados y AWS IoT políticas para conectarse AWS IoT Core y. AWS IoT Greengrass Los certificados y las políticas también permiten AWS IoT Greengrass implementar componentes y configuraciones en los dispositivos principales.

- Dispositivos de cliente

  Los dispositivos cliente de MQTT utilizan certificados y políticas para conectarse al AWS IoT Greengrass servicio AWS IoT Core y al mismo. Esto permite a los dispositivos cliente utilizar la detección AWS IoT Greengrass en la nube para buscar y conectarse a un dispositivo principal de Greengrass. Un dispositivo de cliente utiliza el mismo certificado para conectarse al servicio en la nube de AWS IoT Core y a los dispositivos principales. Los dispositivos de cliente también utilizan información de detección para la autenticación mutua con el dispositivo principal. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Certificados X.509

La comunicación entre los dispositivos principales y los dispositivos cliente y entre los dispositivos AWS IoT Core y/o AWS IoT Greengrass debe estar autenticada. Esta autenticación mutua se basa en certificados de dispositivo X.509 registrados y claves criptográficas.

En un AWS IoT Greengrass entorno, los dispositivos utilizan certificados con claves públicas y privadas para las siguientes conexiones de Transport Layer Security (TLS):

- El componente de AWS IoT cliente del dispositivo principal de Greengrass que se conecta a Internet AWS IoT Core y a AWS IoT Greengrass través de ella.
- Dispositivos cliente que se conectan a AWS IoT Greengrass través de Internet para descubrir los dispositivos principales.
- El componente agente MQTT en el núcleo de Greengrass que se conecta a los dispositivos de Greengrass en el grupo a través de la red local.

AWS IoT Greengrass los dispositivos principales almacenan los certificados en la carpeta raíz de Greengrass.

Certificados de entidad de certificación (CA)

Los dispositivos principales y los dispositivos cliente de Greengrass descargan un certificado de CA raíz que se utiliza para la autenticación con los servicios AWS IoT Core y AWS IoT Greengrass . Le recomendamos que utilice un certificado de entidad de certificación raíz de Amazon Trust Services (ATS), como Amazon Root CA 1. Para obtener más información, consulte Certificados de CA para autenticación de servidor en la Guía del desarrollador de AWS IoT Core .

Los dispositivos de cliente también descargan un certificado de CA del dispositivo principal de Greengrass. Este certificado se utiliza para validar el certificado del servidor MQTT en el dispositivo principal durante la autenticación mutua.

Rotación de certificados en el agente MQTT local

Al habilitar la compatibilidad con dispositivos de cliente, los dispositivos principales de Greengrass generan un certificado de servidor MQTT local que los dispositivos de cliente utilizan para la autenticación mutua. Este certificado está firmado por el certificado de CA del dispositivo principal, que el dispositivo principal almacena en la AWS IoT Greengrass nube. Los dispositivos de cliente recuperan el certificado de CA del dispositivo principal cuando descubren el dispositivo principal.

Utilizan el certificado de CA del dispositivo principal para verificar el certificado del servidor MQTT del dispositivo principal cuando se conectan al dispositivo principal. El certificado de CA del dispositivo principal caduca a los 5 años.

El certificado del servidor MQTT caduca cada 7 días de forma predeterminada y puede configurar esta duración entre 2 y 10 días. Este período limitado se basa en las prácticas recomendadas de seguridad. Esta rotación ayuda a mitigar la amenaza de que un atacante robe el certificado del servidor MQTT y la clave privada para hacerse pasar por el dispositivo principal de Greengrass.

El dispositivo principal de Greengrass rota el certificado del servidor MQTT 24 horas antes de que caduque. El dispositivo principal de Greengrass genera un nuevo certificado y reinicia el agente MQTT local. En este momento, se desconectan todos los dispositivos de cliente conectados al dispositivo principal de Greengrass. Los dispositivos de cliente se pueden volver a conectar al dispositivo principal de Greengrass después de un breve periodo.

AWS IoT políticas para las operaciones del plano de datos

Utilice AWS IoT políticas para autorizar el acceso a los planos de AWS IoT Greengrass datos AWS IoT Core y a los mismos. El plano de datos AWS IoT Core proporciona operaciones para dispositivos, usuarios y aplicaciones. Estas operaciones incluyen la posibilidad de conectarse a los temas AWS IoT Core y suscribirse a ellos. El plano AWS IoT Greengrass de datos proporciona operaciones para los dispositivos Greengrass. Para obtener más información, consulte AWS IoT Greengrass V2 acciones políticas. Estas operaciones incluyen la capacidad de resolver las dependencias de los componentes y descargar artefactos de componentes públicos.

Una AWS IoT política es un documento JSON similar a una política de IAM. Contiene una o varias instrucciones de política que especifican las siguientes propiedades:

- `Effect`. El modo de acceso, que puede ser `Allow` o `Deny`.
- `Action`. La lista de acciones permitidas o denegadas por la política.
- `Resource`. La lista de recursos en los que se permite o deniega la acción.

AWS IoT las políticas * se admiten como caracteres comodín y tratan los caracteres comodín (+y#) de MQTT como cadenas literales. Para obtener más información sobre el * comodín, consulte Uso del comodín en un recurso ARNs en la Guía del usuario.AWS Identity and Access Management

Para obtener más información, consulte Políticas de AWS IoT y Acciones de política de AWS IoT en la Guía del desarrollador de AWS IoT Core .

Important

Las variables de política de objetos (`iot:Connection.Thing.*`) no son compatibles con las políticas de AWS IoT para dispositivos principales ni operaciones del plano de datos de Greengrass. En su lugar, puede utilizar un comodín que haga coincidir varios dispositivos con nombres similares. Por ejemplo, puede especificar `MyGreengrassDevice*` para que coincida con `MyGreengrassDevice1`, `MyGreengrassDevice2`, etc.

Note

AWS IoT Core permite adjuntar AWS IoT políticas a grupos de cosas para definir los permisos para grupos de dispositivos. Las políticas de grupos de cosas no permiten el acceso a las operaciones del plano de AWS IoT Greengrass datos. Para permitir que una cosa acceda a una operación del plano de AWS IoT Greengrass datos, añada el permiso a una AWS IoT política que adjunte al certificado de la cosa.

AWS IoT Greengrass V2 acciones políticas

AWS IoT Greengrass V2 define las siguientes acciones políticas que los dispositivos principales y los dispositivos cliente de Greengrass pueden usar en AWS IoT las políticas. Para especificar un recurso para una acción de política, debe utilizar el nombre de recurso de Amazon (ARN) del recurso.

Acciones del dispositivo principal

`greengrass:GetComponentVersionArtifact`

Conceda permiso para obtener la URL prefirmada para descargar un artefacto de componente público o un artefacto de componente de Lambda.

Este permiso se evalúa cuando un dispositivo principal recibe una implementación que especifica un componente público o de Lambda que tiene artefactos. Si el dispositivo principal ya tiene el artefacto, no lo volverá a descargar.

Tipo de recurso: `componentVersion`

Formato del ARN de recurso: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

Conceda permiso para enumerar los componentes que cumplen los requisitos de componente, versión y plataforma de una implementación. Si los requisitos entran en conflicto o no existen componentes que los cumplan, esta operación devuelve un error y la implementación no se realiza correctamente en el dispositivo.

Este permiso se evalúa cuando un dispositivo principal recibe una implementación que especifica los componentes.

Tipo de recurso: ninguno

Formato del ARN de recurso: *

`greengrass:GetDeploymentConfiguration`

Conceda permiso para obtener una URL prefirmada para descargar un documento de implementación de gran tamaño.

Este permiso se evalúa cuando un dispositivo principal recibe una implementación que especifica un documento de implementación de más de 7 KB (si la implementación se dirige a un objeto) o 31 KB (si la implementación está dirigida a un grupo de objetos). El documento de implementación incluye configuraciones de componentes, políticas de implementación y metadatos de implementación. Para obtener más información, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Esta característica está disponible para la versión 2.3.0 y versiones posteriores del componente núcleo de Greengrass.

Tipo de recurso: ninguno

Formato del ARN de recurso: *

`greengrass:ListThingGroupsForCoreDevice`

Concede permiso para obtener la jerarquía de grupos de objetos de un dispositivo principal.

Este permiso se comprueba cuando un dispositivo principal recibe una implementación desde AWS IoT Greengrass. El dispositivo principal utiliza esta acción para identificar si se ha eliminado de un grupo de objetos desde la última implementación. Si el dispositivo principal se eliminó de un grupo de objetos y ese grupo de objetos es el objetivo de una implementación en el dispositivo principal, el dispositivo principal elimina los componentes instalados por esa implementación.

Esta característica la utilizan la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Tipo de recurso: `thing` (dispositivo principal)

Formato del ARN de recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

Otorga permiso para verificar la identidad de un dispositivo de cliente que se conecta a un dispositivo principal.

Este permiso se evalúa cuando un dispositivo principal ejecuta el componente de autenticación del dispositivo de cliente y recibe una conexión MQTT desde un dispositivo de cliente. El dispositivo de cliente presenta su certificado de dispositivo AWS IoT . A continuación, el dispositivo principal envía el certificado del dispositivo al servicio en la nube de AWS IoT Greengrass para verificar la identidad del dispositivo de cliente. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Tipo de recurso: ninguno

Formato del ARN de recurso: *

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

Otorga permiso para comprobar si un dispositivo de cliente está asociado a un certificado AWS IoT .

Este permiso se evalúa cuando un dispositivo principal ejecuta el componente de autenticación del dispositivo de cliente y autoriza a un dispositivo de cliente a conectarse a través de MQTT. Para obtener más información, consulte Interacción con dispositivos IoT locales.

> **Note**
>
> Para que un dispositivo principal utilice esta operación, la función de servicio Greengrass debe estar asociada a usted Cuenta de AWS y permitir el `iot:DescribeCertificate` permiso.

Tipo de recurso: `thing` (dispositivo de cliente)

Formato del ARN de recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

Otorga permiso para cargar certificados de la autoridad de certificación (CA) que los dispositivos de cliente pueden descargar para verificar el dispositivo principal.

Este permiso se evalúa cuando un dispositivo principal instala y ejecuta el componente de autenticación del dispositivo de cliente. Este componente crea una autoridad de certificados local y utiliza esta operación para cargar sus certificados de CA. Los dispositivos de cliente descargan estos certificados de CA cuando utilizan la operación Discover para encontrar los dispositivos principales a los que pueden conectarse. Cuando los dispositivos de cliente se conectan a un agente MQTT en un dispositivo principal, utilizan estos certificados de CA para verificar la identidad del dispositivo principal. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Tipo de recurso: ninguno

Formato de ARN: *

`greengrass:GetConnectivityInfo`

Concede permiso para obtener información de conectividad para un dispositivo principal. Esta información describe cómo los dispositivos de cliente se pueden conectar al dispositivo principal.

Este permiso se evalúa cuando un dispositivo principal instala y ejecuta el componente de autenticación del dispositivo de cliente. Este componente utiliza la información de conectividad para generar certificados de CA válidos para cargarlos en el servicio AWS IoT Greengrass en la nube junto con la PutCertificateAuthoriesoperación. Los dispositivos de cliente utilizan estos certificados de CA para verificar la identidad del dispositivo principal. Para obtener más información, consulte Interacción con dispositivos IoT locales.

También puede utilizar esta operación en el plano de AWS IoT Greengrass control para ver la información de conectividad de un dispositivo principal. Para obtener más información, consulta GetConnectivityInfo en la AWS IoT Greengrass V1 Referencia de la API de .

Tipo de recurso: `thing` (dispositivo principal)

Formato del ARN de recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:UpdateConnectivityInfo`

Concede permiso para actualizar la información de conectividad de un dispositivo principal. Esta información describe cómo los dispositivos de cliente se pueden conectar al dispositivo principal.

Este permiso se evalúa cuando un dispositivo principal ejecuta el componente detector de IP. Este componente identifica la información que los dispositivos de cliente necesitan para conectarse al dispositivo principal de la red local. A continuación, este componente utiliza esta operación para cargar la información de conectividad en el servicio AWS IoT Greengrass en la nube, de modo que los dispositivos cliente puedan recuperar esta información con la operación Discover. Para obtener más información, consulte Interacción con dispositivos IoT locales.

También puede utilizar esta operación en el plano de AWS IoT Greengrass control para actualizar manualmente la información de conectividad de un dispositivo principal. Para obtener más información, consulta UpdateConnectivityInfo en la AWS IoT Greengrass V1 Referencia de la API de .

Tipo de recurso: `thing` (dispositivo principal)

Formato del ARN de recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

Acciones del dispositivo de cliente

`greengrass:Discover`

Concede permiso para descubrir la información de conectividad de los dispositivos principales a los que se puede conectar un dispositivo de cliente. Esta información describe cómo el dispositivo de cliente se puede conectar a los dispositivos principales. Un dispositivo cliente solo puede detectar los dispositivos principales a los que esté asociado mediante esta BatchAssociateClientDeviceWithCoreDeviceoperación. Para obtener más información, consulte Interacción con dispositivos IoT locales.

Tipo de recurso: `thing` (dispositivo de cliente)

Formato del ARN de recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

Actualice la AWS IoT política de un dispositivo principal

Puedes usar las AWS IoT consolas AWS IoT Greengrass y la AWS IoT API para ver y actualizar la AWS IoT política de un dispositivo principal.

> **Note**
>
> Si has utilizado el instalador de software AWS IoT Greengrass principal para aprovisionar recursos, tu dispositivo principal tiene una AWS IoT política que permite el acceso a todas AWS IoT Greengrass las acciones (`greengrass:*`). Puede seguir estos pasos para restringir el acceso únicamente a las acciones que utiliza un dispositivo principal.

Revisa y actualiza la AWS IoT política de un dispositivo principal (consola)

1. En el menú de navegación de la consola de AWS IoT Greengrass, elija Dispositivos principales.
2. En la página Dispositivos principales, elija el dispositivo principal que desea actualizar.
3. En la página de detalles del dispositivo principal, elija el enlace al Objeto del dispositivo principal. Este enlace abre la página de detalles del objeto en la consola de AWS IoT .
4. En la página de detalles del objeto, elija Certificados.
5. En la pestaña Certificados, elija el certificado activo del objeto.
6. En la página de detalles del certificado, elija Políticas.
7. En la pestaña Políticas, selecciona la AWS IoT política que deseas revisar y actualizar. Puede agregar los permisos necesarios a cualquier política que esté asociada al certificado activo del dispositivo principal.

   > **Note**
   >
   > Si ha utilizado el instalador de software AWS IoT Greengrass principal para aprovisionar recursos, tiene dos AWS IoT políticas. Le recomendamos que elija la política denominada GreengrassV2IoTThingPolicy, si existe. Los dispositivos principales que cree con el instalador rápido usan este nombre de política de forma predeterminada. Si agrega permisos a esta política, también los otorga a otros dispositivos principales que usan esta política.

8. En la descripción general de la política, elija Editar la versión activa.

9. Revise la política y agregue, elimine o edite los permisos según sea necesario.
10. Para establecer una nueva versión de la política como la versión activa, en Estado de la versión de la política, seleccione Establecer la versión editada como la versión activa de esta política.
11. Seleccione Guardar como versión nueva.

Revise y actualice la AWS IoT política de un dispositivo principal (AWS CLI)

1. Enumere los principios del AWS IoT dispositivo principal. Las entidades principales del objeto pueden ser certificados de dispositivo X.509 u otros identificadores. Ejecute el siguiente comando y *MyGreengrassCore* sustitúyalo por el nombre del dispositivo principal.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   La operación devuelve una respuesta que enumera las entidades principales del objeto del dispositivo principal.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. Identifique el certificado activo del dispositivo principal. Ejecute el siguiente comando y *certificateId* sustitúyalo por el ID de cada certificado del paso anterior hasta que encuentre el certificado activo. El ID del certificado es la cadena hexadecimal que se encuentra al final del ARN del certificado. El argumento `--query` especifica que solo se muestre el estado del certificado.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   La operación devuelve el estado del certificado en forma de cadena. Por ejemplo, si el certificado está activo, la operación muestra `"ACTIVE"`.

3. Enumere las AWS IoT políticas que se adjuntan al certificado. Ejecute el siguiente comando y reemplace ARN del certificado por el ARN del certificado.

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

La operación devuelve una respuesta en la que se enumeran AWS IoT las políticas adjuntas al certificado.

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. Elija la política que desee ver y actualizar.

> **Note**
>
> Si utilizó el instalador de software AWS IoT Greengrass principal para aprovisionar recursos, tiene dos AWS IoT políticas. Le recomendamos que elija la política denominada GreengrassV2IoTThingPolicy, si existe. Los dispositivos principales que cree con el instalador rápido usan este nombre de política de forma predeterminada. Si agrega permisos a esta política, también los otorga a otros dispositivos principales que usan esta política.

5. Obtenga el documento de la política. Ejecute el siguiente comando y *GreengrassV2IoTThingPolicy* sustitúyalo por el nombre de la política.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

La operación devuelve una respuesta que contiene el documento de política y otra información sobre la política. El documento de política es un objeto JSON serializado como una cadena.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17       \\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Use un conversor en línea u otra herramienta para convertir la cadena del documento de la política en un objeto JSON y, a continuación, guárdela en un archivo denominado `iot-policy.json`.

   Por ejemplo, si tiene instalada la herramienta jq, puede ejecutar el siguiente comando para obtener el documento de la política, convertirlo en un objeto JSON y guardar el documento de la política como un objeto JSON.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. Revise el documento de política y agregue, elimine o edite los permisos según sea necesario.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano para abrir el archivo.

   ```
   nano iot-policy.json
   ```

   Cuando haya terminado, el documento de política podría tener un aspecto similar a la AWS IoT política mínima para los dispositivos principales.

8. Guarde los cambios como una nueva versión de la política. Ejecute el siguiente comando y *GreengrassV2IoTThingPolicy* sustitúyalo por el nombre de la política.

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   Si se realiza correctamente, la operación devuelve una respuesta similar a la del siguiente ejemplo.

   ```
   {
       "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy",
       "policyDocument": "{\
     \\"Version\\": \\"2012-10-17       \\",\
     \\"Statement\\": [\
       {\
         \\"Effect\\": \\"Allow\\",\
         \\"Action\\": [\
   \\t\\t\\"iot:Connect\\",\
   \\t\\t\\"iot:Publish\\",\
   \\t\\t\\"iot:Subscribe\\",\
   \\t\\t\\"iot:Receive\\",\
   \\t\\t\\"greengrass:*\\"\
         ],\
         \\"Resource\\": \\"*\\"\
       }\
     ]\
   }",
       "policyVersionId": "2",
       "isDefaultVersion": true
   }
   ```

AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales

Important

Las versiones posteriores del componente núcleo de Greengrass requieren permisos adicionales en la política mínima AWS IoT . Es posible que tenga que actualizar las políticas de AWS IoT de sus dispositivos principales para conceder permisos adicionales.

- Los dispositivos principales que ejecutan la versión 2.5.0 y versiones posteriores del núcleo de Greengrass utilizan el permiso `greengrass:ListThingGroupsForCoreDevice` para desinstalar componentes al eliminar un dispositivo principal de un grupo de objetos.
- Los dispositivos principales que ejecutan la versión 2.3.0 y versiones posteriores del núcleo de Greengrass utilizan el permiso `greengrass:GetDeploymentConfiguration` para admitir documentos de configuración de implementación de gran tamaño.

La siguiente política de ejemplo incluye un conjunto mínimo de acciones necesario para respaldar la funcionalidad básica de Greengrass para su dispositivo del núcleo.

- La política de `Connect` incluye un comodín `*` después del nombre del objeto del dispositivo principal (por ejemplo, *`core-device-thing-name`*`*`). El dispositivo principal utiliza el mismo certificado de dispositivo para realizar múltiples suscripciones simultáneas AWS IoT Core, pero es posible que el ID de cliente de una conexión no coincida exactamente con el nombre del dispositivo principal. Después de las primeras 50 suscripciones, el dispositivo principal utiliza *`core-device-thing-name`*`#`*`number`* como ID de cliente, donde incrementa *`number`* por cada 50 suscripciones adicionales. Por ejemplo, cuando un dispositivo principal denominado `MyCoreDevice` crea 150 suscripciones simultáneas, utiliza el siguiente cliente: IDs
  - Suscripciones del 1 al 50: `MyCoreDevice`
  - Suscripciones del 51 al 100: `MyCoreDevice#2`
  - Suscripciones 101 a 150: `MyCoreDevice#3`

  El comodín permite que el dispositivo principal se conecte cuando utiliza estos clientes IDs que tienen un sufijo.

- La política muestra los temas de MQTT y los filtros de tema en los que el dispositivo del núcleo puede publicar mensajes, suscribirse y recibir mensajes, incluidos temas utilizados para estado de sombra. Para permitir el intercambio de mensajes entre AWS IoT Core los componentes de Greengrass y los dispositivos cliente, especifique los temas y los filtros de temas que desee permitir. Para obtener más información, consulte Ejemplos de política de publicación/suscripción en la Guía del desarrollador de AWS IoT Core .
- La política concede permiso para publicar datos de telemetría en el siguiente tema.

  ```
  $aws/things/core-device-thing-name/greengrass/health/json
  ```

  Puede eliminar este permiso en los dispositivos principales en los que se deshabilita la telemetría. Para obtener más información, consulte Recopile datos de telemetría del estado del sistema de los dispositivos principales AWS IoT Greengrass.
- La política concede permiso para asumir una función de IAM mediante un alias de AWS IoT función. El dispositivo principal utiliza esta función, denominada función de intercambio de fichas, para adquirir AWS credenciales que puede utilizar para autenticar las solicitudes. AWS Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

  Al instalar el software AWS IoT Greengrass principal, se crea y se adjunta una segunda AWS IoT política que incluye solo este permiso. Si incluye este permiso en la AWS IoT política principal de su dispositivo principal, puede separar y eliminar la otra AWS IoT política.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/core-device-
thing-name*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/greengrass/health/json",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/greengrassv2/health/json",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/jobs/*",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name/jobs/*",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/token-
exchange-role-alias-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

AWS IoT Política mínima de compatibilidad con los dispositivos cliente

El siguiente ejemplo de política incluye el conjunto mínimo de acciones necesarias para permitir la interacción con los dispositivos de cliente en un dispositivo principal. Para ser compatible con los dispositivos cliente, un dispositivo principal debe tener los permisos de esta AWS IoT política además de la AWS IoT política mínima para su funcionamiento básico.

- La política permite que el dispositivo principal actualice su propia información de conectividad. Este permiso (`greengrass:UpdateConnectivityInfo`) solo es necesario si se implementa el componente del detector de IP en el dispositivo principal.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name-gci/shadow/update/delta",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
device-thing-name-gci/shadow/get/accepted"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/update/delta",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-
device-thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/core-device-thing-name"
            ]
        }
    ]
}
```

AWS IoT Política mínima para los dispositivos cliente

El siguiente ejemplo de política incluye el conjunto mínimo de acciones necesarias para que un dispositivo de cliente detecte los dispositivos principales a los que se conecta y se comunica a través de MQTT. La AWS IoT política del dispositivo cliente debe incluir la `greengrass:Discover` acción que permita al dispositivo descubrir la información de conectividad de sus dispositivos principales Greengrass asociados. En la sección `Resource`, especifique el nombre de recurso de Amazon (ARN) del dispositivo de cliente, no el ARN del dispositivo principal de Greengrass.

- La política permite la comunicación en todos los temas de MQTT. Para seguir las prácticas recomendadas de seguridad, restrinja los permisos `iot:Publish`, `iot:Subscribe` y `iot:Receive` al conjunto mínimo de temas que un dispositivo de cliente requiera para su caso de uso.
- La política permite que el dispositivo descubra los dispositivos principales para AWS IoT todo tipo de dispositivos. Para seguir las mejores prácticas de seguridad, restrinja el `greengrass:Discover` permiso al dispositivo cliente o AWS IoT a un comodín que coincida con un conjunto de AWS IoT elementos.

  > **Important**
  >
  > Las variables de política de objetos (`iot:Connection.Thing.*`) no son compatibles con las políticas de AWS IoT para dispositivos principales ni operaciones del plano de datos de Greengrass. En su lugar, puede utilizar un comodín que haga coincidir varios dispositivos con nombres similares. Por ejemplo, puede especificar `MyGreengrassDevice*` para que coincida con `MyGreengrassDevice1`, `MyGreengrassDevice2`, etc.

- La AWS IoT política de un dispositivo cliente no suele requerir permisos ni `iot:DeleteThingShadow` acciones `iot:GetThingShadowiot:UpdateThingShadow`, ya que el dispositivo principal de Greengrass gestiona las operaciones de sincronización oculta para los dispositivos cliente. Para permitir que el dispositivo principal gestione las sombras de los dispositivos cliente, compruebe que la AWS IoT política del dispositivo principal permita estas acciones y que la `Resource` sección incluya ARNs los dispositivos cliente.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:Discover"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/*"
            ]
        }
    ]
}
```

Administración de identidad y acceso para AWS IoT Greengrass

AWS Identity and Access Management (IAM) es una herramienta Servicio de AWS que ayuda al administrador a controlar de forma segura el acceso a los AWS recursos. Los administradores de IAM controlan quién puede autenticarse (iniciar sesión) y quién puede autorizarse (tener permisos) para usar los recursos. AWS IoT Greengrass La IAM es una Servicio de AWS opción que puede utilizar sin coste adicional.

> **Note**
>
> En este tema se describen conceptos y características de IAM. Para obtener información sobre las funciones de IAM compatibles con AWS IoT Greengrass, consulte. the section called “Cómo AWS IoT Greengrass funciona con IAM”

Público

La forma de utilizar AWS Identity and Access Management (IAM) varía en función de la función que desempeñe:

- Usuario del servicio: solicite permisos al administrador si no puede acceder a las características (consulte Solución de problemas de identidad y acceso para AWS IoT Greengrass).
- Administrador del servicio: determine el acceso de los usuarios y envíe las solicitudes de permiso (consulte Cómo AWS IoT Greengrass funciona con IAM).
- Administrador de IAM: escribe las políticas para administrar el acceso (consulte Ejemplos de políticas basadas en la identidad para AWS IoT Greengrass).

Autenticación con identidades

La autenticación es la forma en que inicias sesión AWS con tus credenciales de identidad. Debe autenticarse como usuario de Usuario raíz de la cuenta de AWS IAM o asumir una función de IAM.

Puede iniciar sesión como una identidad federada con las credenciales de una fuente de identidad, como AWS IAM Identity Center (IAM Identity Center), la autenticación de inicio de sesión único o las credenciales. Google/Facebook Para obtener más información sobre el inicio de sesión, consulte Cómo iniciar sesión en la Cuenta de AWS en la Guía del usuario de AWS Sign-In .

Para el acceso programático, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte AWS Signature Version 4 para solicitudes de API en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear un Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario Cuenta de AWS raíz que tiene acceso completo a todos Servicios de AWS los recursos. Se recomiendaencarecidamente que no utilice el usuario raíz para las tareas diarias. Para ver la lista completa de las tareas que requieren credenciales de usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Usuarios y grupos de IAM

Un usuario de IAM es una identidad con permisos específicos para una sola persona o aplicación. Recomendamos el uso de credenciales temporales en lugar de usuarios de IAM con credenciales de larga duración. Para obtener más información, consulte Exigir a los usuarios humanos que utilicen la federación con un proveedor de identidad para acceder AWS mediante credenciales temporales en la Guía del usuario de IAM.

Un grupo de IAM especifica un conjunto de usuarios de IAM y facilita la administración de los permisos para grupos grandes de usuarios. Para obtener más información, consulte Casos de uso para usuarios de IAM en la Guía del usuario de IAM.

Roles de IAM

Un Rol de IAM es una identidad con permisos específicos que proporciona credenciales temporales. Puede asumir un rol cambiando de un rol de usuario a uno de IAM (consola) o llamando a una AWS CLI operación de AWS API. Para obtener más información, consulte Métodos para asumir un rol en la Guía del usuario de IAM.

Los roles de IAM son útiles para el acceso de usuario federado, los permisos de usuario de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon EC2. Para obtener más información, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Administración del acceso con políticas

AWS Para controlar el acceso, puede crear políticas y adjuntarlas a AWS identidades o recursos. Una política define los permisos cuando están asociados a una identidad o un recurso. AWS evalúa estas políticas cuando un director hace una solicitud. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre los documentos de políticas de JSON, consulte Información general de políticas de JSON en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en la identidad, consulte Definición de permisos de IAM personalizados con políticas administradas por el cliente en la Guía del usuario de IAM.

Las políticas basadas en identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte Selección entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de políticas JSON que se asocian a un recurso. Los ejemplos incluyen las Políticas de confianza de roles de IAM y las Políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Debe especificar una entidad principal en una política basada en recursos.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No puedes usar políticas AWS gestionadas de IAM en una política basada en recursos.

Listas de control de acceso () ACLs

Las listas de control de acceso (ACLs) controlan qué responsables (miembros de la cuenta, usuarios o roles) tienen permisos para acceder a un recurso. ACLs son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3 y Amazon VPC son ejemplos de servicios compatibles. AWS WAF ACLs Para obtener más información ACLs, consulte la descripción general de la lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite tipos de políticas adicionales que pueden establecer los permisos máximos otorgados por los tipos de políticas más comunes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCPs): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte Políticas de control de servicios en la Guía del usuario de AWS Organizations .
- Políticas de control de recursos (RCPs): establece los permisos máximos disponibles para los recursos de tus cuentas. Para obtener más información, consulte Políticas de control de recursos (RCPs) en la Guía del AWS Organizations usuario.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para saber cómo se AWS determina si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la lógica de evaluación de políticas en la Guía del usuario de IAM.

Véase también

- the section called “Cómo AWS IoT Greengrass funciona con IAM”
- the section called “Ejemplos de políticas basadas en identidades”
- the section called “Solución de problemas de identidades y accesos”

Cómo AWS IoT Greengrass funciona con IAM

Antes de usar IAM para administrar el acceso AWS IoT Greengrass, debe comprender las funciones de IAM que puede utilizar. AWS IoT Greengrass

Característica de IAM	¿Compatible con Greengrass?
Políticas basadas en identidad con permisos de nivel de recursos	Sí
Políticas basadas en recursos	No
Listas de control de acceso () ACLs	No
Autorización basada en etiquetas	Sí
Credenciales temporales	Sí
Roles vinculados al servicio	No
Roles de servicio	Sí

Para obtener una visión general de cómo funcionan otros AWS servicios con IAM, consulte AWS los servicios que funcionan con IAM en la Guía del usuario de IAM.

Políticas basadas en la identidad para AWS IoT Greengrass

Con las políticas de IAM basadas en la identidad, puede especificar las acciones y los recursos permitidos o denegados y las condiciones en las que se permiten o deniegan las acciones. AWS IoT Greengrass admite acciones, recursos y claves de condición específicos. Para obtener más información acerca de los elementos que utiliza en una política, consulte Referencia de los elementos de las políticas de JSON de IAM en la Guía del usuario de IAM.

Acciones

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Acciones políticas para AWS IoT Greengrass usar el `greengrass:` prefijo antes de la acción. Por ejemplo, para permitir que alguien utilice la operación de la `ListCoreDevices` API para enumerar sus dispositivos principales Cuenta de AWS, debes incluir la `greengrass:ListCoreDevices` acción en su política. Las declaraciones de política deben incluir un `NotAction` elemento `Action` o. AWS IoT Greengrass define su propio conjunto de acciones que describen las tareas que puede realizar con este servicio.

Para especificar varias acciones en una misma instrucción, inclúyalas entre corchetes (`[ ]`) y sepárelas por comas, tal y como se indica a continuación:

```
"Action": [
  "greengrass:action1",
  "greengrass:action2",
  "greengrass:action3"
]
```

Puede utilizar comodines (*) para especificar varias acciones. Por ejemplo, para especificar todas las acciones que comiencen con la palabra `List`, incluya la siguiente acción:

```
"Action": "greengrass:List*"
```

> **Note**
>
> Se recomienda evitar el uso de comodines para especificar todas las acciones disponibles para un servicio. Como práctica recomendada, debe conceder permisos de mínimo privilegio y acotar el alcance de los permisos en una política. Para obtener más información, consulte the section called “Conceda los mínimos permisos posibles”.

Para ver la lista completa de AWS IoT Greengrass acciones, consulte las acciones definidas por AWS IoT Greengrass en la Guía del usuario de IAM.

Recursos

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). En el caso de las acciones que no admiten permisos por recurso, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

La siguiente tabla contiene el AWS IoT Greengrass recurso ARNs que se puede utilizar como `Resource` elemento de una declaración de política. Para ver un mapeo de los permisos a nivel de recursos admitidos para AWS IoT Greengrass las acciones, consulte las acciones definidas por AWS IoT Greengrass en la Guía del usuario de IAM.

Algunas AWS IoT Greengrass acciones (por ejemplo, algunas operaciones de lista) no se pueden realizar en un recurso específico. En dichos casos, debe utilizar solo el carácter comodín.

```
"Resource": "*"
```

Para especificar varios recursos ARNs en una sentencia, enumérelos entre corchetes (`[]`) y sepárelos con comas, de la siguiente manera:

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

Para obtener más información sobre los formatos ARN, consulte Nombres de recursos de Amazon (ARNs) y espacios de nombres AWS de servicios en. Referencia general de Amazon Web Services

Claves de condición

Los administradores pueden usar las políticas de AWS JSON para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` especifica cuándo se ejecutan las instrucciones en función de criterios definidos. Puede crear expresiones condicionales que utilizan operadores de condición, tales como

igual o menor que, para que la condición de la política coincida con los valores de la solicitud. Para ver todas las claves de condición AWS globales, consulte las claves de contexto de condición AWS globales en la Guía del usuario de IAM.

Ejemplos

Para ver ejemplos de políticas AWS IoT Greengrass basadas en la identidad, consulte. the section called “Ejemplos de políticas basadas en identidades”

Políticas basadas en recursos para AWS IoT Greengrass

AWS IoT Greengrass no admite políticas basadas en recursos.

Listas de control de acceso () ACLs

AWS IoT Greengrass no admite ACLs.

Autorización basada en AWS IoT Greengrass etiquetas

Puede adjuntar etiquetas a AWS IoT Greengrass los recursos compatibles o pasarlas en una solicitud AWS IoT Greengrass. Para controlar el acceso utilizando etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` o `aws:TagKeys`. Para obtener más información, consulte Etiquetar los recursos.

Funciones de IAM para AWS IoT Greengrass

Un rol de IAM es una entidad de la Cuenta de AWS que dispone de permisos específicos.

Usar credenciales temporales con AWS IoT Greengrass

Las credenciales temporales se utilizan para iniciar sesión con federación, adoptar un rol de IAM o adoptar un rol de acceso entre cuentas. Las credenciales de seguridad temporales se obtienen mediante una llamada a operaciones de la API de AWS STS , como AssumeRole o GetFederationToken.

En el núcleo de Greengrass, las credenciales temporales para el rol de dispositivo están disponibles para los componentes de Greengrass. Si sus componentes usan el AWS SDK, no necesita agregar lógica para obtener las credenciales, ya que el AWS SDK lo hace por usted.

Roles vinculados a servicios

AWS IoT Greengrass no admite funciones vinculadas a servicios.

Roles de servicio

Esta característica permite que un servicio asuma un rol de servicio en su nombre. Este rol permite que el servicio obtenga acceso a los recursos de otros servicios para completar una acción en su nombre. Los roles de servicio aparecen en su cuenta de IAM y son propiedad de la cuenta. Esto significa que un administrador de IAM puede cambiar los permisos de este rol. Sin embargo, hacerlo podría deteriorar la funcionalidad del servicio.

AWS IoT Greengrass los dispositivos principales utilizan una función de servicio para permitir que los componentes de Greengrass y las funciones de Lambda accedan a algunos de sus AWS recursos en su nombre. Para obtener más información, consulte the section called “Autorizar a los dispositivos principales a interactuar con AWS los servicios”.

AWS IoT Greengrass utiliza un rol de servicio para acceder a algunos de sus AWS recursos en su nombre. Para obtener más información, consulte Rol de servicio de Greengrass.

Ejemplos de políticas basadas en la identidad para AWS IoT Greengrass

De forma predeterminada, los usuarios y los roles de IAM no tienen permiso para crear, ver ni modificar recursos de AWS IoT Greengrass . Tampoco pueden realizar tareas con la API Consola de administración de AWS AWS CLI, o AWS . Un administrador de IAM debe crear políticas de IAM que concedan permisos a los usuarios y a los roles para realizar operaciones de la API concretas en los recursos especificados que necesiten. El administrador debe asociar esas políticas a los usuarios o grupos de IAM que necesiten esos permisos.

Prácticas recomendadas sobre las políticas

Las políticas basadas en la identidad determinan si alguien puede crear AWS IoT Greengrass recursos de tu cuenta, acceder a ellos o eliminarlos. Estas acciones pueden generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas AWS administradas y avance hacia los permisos con privilegios mínimos: para empezar a conceder permisos a sus usuarios y cargas de trabajo, utilice las políticas AWS administradas que otorgan permisos para muchos casos de uso comunes. Están disponibles en su. Cuenta de AWS Le recomendamos que reduzca aún más los permisos

definiendo políticas administradas por el AWS cliente que sean específicas para sus casos de uso. Con el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de tarea en la Guía de usuario de IAM.

- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte Políticas y permisos en IAM en la Guía del usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puedes usar condiciones para conceder el acceso a las acciones del servicio si se utilizan a través de una acción específica Servicio de AWS, por ejemplo CloudFormation. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.
- Utiliza el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte Validación de políticas con el Analizador de acceso de IAM en la Guía del usuario de IAM.
- Requerir autenticación multifactor (MFA): si tiene un escenario que requiere usuarios de IAM o un usuario raíz en Cuenta de AWS su cuenta, active la MFA para mayor seguridad. Para exigir la MFA cuando se invoquen las operaciones de la API, añada condiciones de MFA a sus políticas. Para más información, consulte Acceso seguro a la API con MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Ejemplos de políticas

En el ejemplo siguiente, las políticas definidas por el cliente conceden permisos para situaciones comunes.

Ejemplos

- Cómo permitir a los usuarios consultar sus propios permisos

Para obtener más información acerca de cómo crear una política basada en identidad de IAM con estos documentos de políticas de JSON de ejemplo, consulte Creación de políticas en la pestaña JSON en la Guía del usuario de IAM.

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para completar esta acción en la consola o mediante programación mediante la API o. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

Autorizar a los dispositivos principales a interactuar con AWS los servicios

AWS IoT Greengrass los dispositivos principales utilizan el proveedor de AWS IoT Core credenciales para autorizar las llamadas a AWS los servicios. El proveedor de AWS IoT Core credenciales permite a los dispositivos utilizar sus certificados X.509 como identidad única del dispositivo para autenticar las solicitudes. AWS Esto elimina la necesidad de almacenar un identificador de clave de AWS acceso y una clave de acceso secreta en los dispositivos AWS IoT Greengrass principales. Para obtener más información, consulte Autorizar llamadas directas a AWS los servicios en la Guía para AWS IoT Core desarrolladores.

Al ejecutar el software AWS IoT Greengrass principal, puede optar por aprovisionar los AWS recursos que requiere el dispositivo principal. Esto incluye la función AWS Identity and Access Management (IAM) que asume su dispositivo principal a través del proveedor de AWS IoT Core credenciales. Utilice el `--provision true` argumento para configurar una función y políticas que permitan al dispositivo principal obtener AWS credenciales temporales. Este argumento también configura un alias de AWS IoT rol que apunta a este rol de IAM. Puede especificar el nombre del rol de IAM y el alias del rol que se van a AWS IoT utilizar. Si especifica `--provision true` sin estos otros parámetros de nombre, el dispositivo principal de Greengrass crea y utiliza los siguientes recursos predeterminados:

- Rol de IAM: `GreengrassV2TokenExchangeRole`

  Este rol tiene una política denominada `GreengrassV2TokenExchangeRoleAccess` y una relación de confianza que permite a `credentials.iot.amazonaws.com` asumir el rol. La política incluye los permisos mínimos para el dispositivo principal.

  > **Important**
  >
  > Esta política no incluye el acceso a los archivos en los buckets de S3. Debe agregar permisos al rol para permitir que los dispositivos principales recuperen los artefactos de los componentes de los buckets de S3. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.

- AWS IoT alias del rol: `GreengrassV2TokenExchangeRoleAlias`

Este alias del rol hace referencia al rol de IAM.

Para obtener más información, consulte Paso 3: Instalar el software AWS IoT Greengrass principal.

También puede establecer el alias del rol para un dispositivo principal existente. Para ello, configure el parámetro de configuración `iotRoleAlias` del componente del núcleo de Greengrass.

Puede adquirir AWS credenciales temporales para este rol de IAM a fin de realizar AWS operaciones en sus componentes personalizados. Para obtener más información, consulte Interacción con servicios de AWS.

Temas

- Permisos de rol de servicio para dispositivos principales
- Cómo permitir el acceso a los buckets de S3 para los artefactos del componente

Permisos de rol de servicio para dispositivos principales

El rol permite que el siguiente servicio asuma el rol:

- `credentials.iot.amazonaws.com`

Si utilizas el software AWS IoT Greengrass Core para crear este rol, este utilizará la siguiente política de permisos para permitir que los dispositivos principales se conecten y envíen registros a AWS ellos. El nombre de la política se establece de forma predeterminada en el nombre del rol de IAM que termina en `Access`. Por ejemplo, si usa el nombre de rol de IAM predeterminado, el nombre de esta política es `GreengrassV2TokenExchangeRoleAccess`.

Greengrass nucleus v2.5.0 and later

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Cómo permitir el acceso a los buckets de S3 para los artefactos del componente

El rol de dispositivo principal predeterminado no permite que los dispositivos principales accedan a los buckets de S3. Para implementar componentes que tienen artefactos en buckets de S3, debe agregar el permiso `s3:GetObject` que permita a los dispositivos principales descargar artefactos del componente. Puede agregar una nueva política al rol de dispositivo principal para conceder este permiso.

Adición de una política que permita el acceso a los artefactos del componente en Amazon S3

1. Cree un archivo llamado `component-artifact-policy.json` y copie el siguiente JSON en el archivo. Esta política permite el acceso a todos los archivos en un bucket de S3. Reemplace amzn-s3-demo-bucket por el nombre del bucket de S3 para permitir el acceso del dispositivo principal.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
   ```

```
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

2. Ejecute el siguiente comando para crear la política del documento de política en `component-artifact-policy.json`.

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name MyGreengrassV2ComponentArtifactPolicy \
     --policy-document file://component-artifact-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name MyGreengrassV2ComponentArtifactPolicy ^
     --policy-document file://component-artifact-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name MyGreengrassV2ComponentArtifactPolicy `
     --policy-document file://component-artifact-policy.json
   ```

   Copie la política del nombre de recurso de Amazon (ARN) de la política de los metadatos de salida. Utilice este ARN para asociar la política al rol del dispositivo principal en el siguiente paso.

3. Ejecute el siguiente comando para asociar la política al rol del dispositivo principal. *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre del rol que especificó al ejecutar el software AWS IoT Greengrass Core. Luego, sustituya el ARN de la política por el ARN del paso anterior.

Linux or Unix

```
aws iam attach-role-policy \
  --role-name GreengrassV2TokenExchangeRole \
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Si el comando no tiene ningún resultado, se ha realizado correctamente y su dispositivo principal puede acceder a los artefactos que cargue en este bucket de S3.

Política de IAM mínima para que el instalador aprovisione recursos

Al instalar el software AWS IoT Greengrass Core, puede aprovisionar AWS los recursos necesarios, como una AWS IoT cosa y una función de IAM para su dispositivo. También puede implementar herramientas de desarrollo local en el dispositivo. El instalador necesita AWS credenciales para poder realizar estas acciones en su ordenador Cuenta de AWS. Para obtener más información, consulte Instalación del software AWS IoT Greengrass Core.

El siguiente ejemplo de política incluye el conjunto mínimo de acciones que el instalador necesita para aprovisionar estos recursos. Estos permisos son necesarios si se especifica el argumento `--provision` del instalador. *`account-id`*Sustitúyala por tu Cuenta de AWS ID y *`GreengrassV2TokenExchangeRole`* sustitúyela por el nombre de la función de intercambio de fichas que especifiques con el argumento del `--tes-role-name` instalador.

> **Note**
>
> La declaración de la política `DeployDevTools` solo es necesaria si se especifica el argumento del instalador `--deploy-dev-tools`.

Greengrass nucleus v2.5.0 and later

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [

 "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",

 "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
```

```
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Earlier than v2.5.0

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
```

```
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
            "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

Rol de servicio de Greengrass

La función de servicio de Greengrass es una función de servicio AWS Identity and Access Management (IAM) que autoriza el acceso AWS IoT Greengrass a los recursos de los AWS servicios en su nombre. Esta función permite verificar la identidad de AWS IoT Greengrass los dispositivos cliente y administrar la información de conectividad principal de los dispositivos.

> **Note**
>
> AWS IoT Greengrass V1 también utiliza esta función para realizar tareas esenciales. Para obtener más información, consulte Rol de servicio de Greengrass en la Guía para desarrolladores de AWS IoT Greengrass V1 .

Para permitir el acceso AWS IoT Greengrass a sus recursos, la función de servicio de Greengrass debe estar asociada a la suya Cuenta de AWS y especificarse AWS IoT Greengrass como entidad de confianza. El rol debe incluir la política AWSGreengrassResourceAccessRolePolicyadministrada o una política personalizada que defina permisos equivalentes para las AWS IoT Greengrass funciones que utilice. AWS mantiene esta política, que define el conjunto de permisos que se AWS IoT Greengrass utilizan para acceder a AWS los recursos. Para obtener más información, consulte AWS política gestionada: AWSGreengrass ResourceAccessRolePolicy.

Puedes reutilizar la misma función de servicio de Greengrass en todas partes Regiones de AWS, pero debes asociarla a tu cuenta en todos los Región de AWS lugares donde la utilices. AWS IoT Greengrass Si la función de servicio no está configurada en la versión actual Región de AWS, los dispositivos principales no pueden verificar los dispositivos cliente ni actualizar la información de conectividad.

En las siguientes secciones se describe cómo crear y administrar el rol de servicio de Greengrass con o. Consola de administración de AWS AWS CLI

Temas

- Administración del rol de servicio de Greengrass (consola)
- Administración del rol de servicio de Greengrass (CLI)
- Véase también

> **Note**
>
> Además del rol de servicio que autoriza el acceso de nivel de servicio, puede asignar un rol de intercambio de tokens a los dispositivos principales de Greengrass. La función de intercambio de fichas es una función de IAM independiente que controla la forma en que los componentes de Greengrass y las funciones de Lambda del dispositivo principal pueden acceder a los servicios. AWS Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Administración del rol de servicio de Greengrass (consola)

La AWS IoT consola facilita la administración de su función de servicio de Greengrass. Por ejemplo, al configurar la detección de dispositivos de cliente para un dispositivo principal, la consola comprueba si su Cuenta de AWS está asociada a un rol de servicio de Greengrass en la Región de AWS actual. De lo contrario, la consola puede crear y configurar un rol de servicio por usted. Para obtener más información, consulte the section called “Creación del rol de servicio de Greengrass”.

Puede utilizar la consola de para las siguientes tareas de administración de roles:

Temas

- Buscar el rol de servicio de Greengrass (consola)
- Creación del rol de servicio de Greengrass (consola)
- Cambiar el rol de servicio de Greengrass (consola)
- Desasociar el rol de servicio de Greengrass (consola)

> **Note**
>
> El usuario que ha iniciado sesión en la consola debe tener permisos para ver, crear o cambiar el rol de servicio.

Buscar el rol de servicio de Greengrass (consola)

Siga los siguientes pasos para encontrar el rol de servicio que AWS IoT Greengrass utiliza en el actual Región de AWS.

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación, seleccione Configuración.
3. Desplácese hasta la sección Greengrass service role (Rol de servicio de Greengrass) para ver el rol de servicio y sus políticas.

   Si no ve ningún rol de servicio, la consola puede crear o configurar uno por usted. Para obtener más información, consulte Creación del rol de servicio de Greengrass.

Creación del rol de servicio de Greengrass (consola)

La consola puede crear y configurar un rol de servicio de Greengrass predeterminado por usted. Este rol incluye las siguientes propiedades.

Propiedad	Valor
Name	`Greengrass_ServiceRole`
Entidad de confianza	`AWS service: greengrass`
Política	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Si crea este rol con el script de configuración del dispositivo de AWS IoT Greengrass V1, el nombre del rol es `GreengrassServiceRole_`*`random-string`*.

Al configurar la detección de dispositivos cliente para un dispositivo principal, la consola comprueba si una función de servicio de Greengrass está asociada a la suya Cuenta de AWS en la versión actual. Región de AWS De lo contrario, la consola le solicitará que permita AWS IoT Greengrass leer y escribir en AWS los servicios en su nombre.

Si concede permiso, la consola comprueba si existe un rol denominado `Greengrass_ServiceRole` en su Cuenta de AWS.

- Si la función existe, la consola le asigna la función de servicio a la suya Cuenta de AWS en la actual. Región de AWS
- Si el rol no existe, la consola crea un rol de servicio de Greengrass predeterminado y lo adjunta al tuyo Cuenta de AWS en el actual. Región de AWS

> **Note**
>
> Si desea crear un rol de servicio con políticas de rol personalizadas, utilice la consola de IAM para crear o modificar el rol. Para obtener más información, consulte Crear un rol para delegar permisos a un AWS servicio o Modificar un rol en la Guía del usuario de IAM. Asegúrese de que el rol concede permisos equivalentes a la política administrada de `AWSGreengrassResourceAccessRolePolicy` para las características y recursos que utiliza. Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención de la sustitución confusa entre servicios. Si crea un rol de servicio, vuelva a la AWS IoT consola y asocie el rol al suyo Cuenta de AWS. Puede hacerlo en el rol de servicio de Greengrass en la página Configuración.

Cambiar el rol de servicio de Greengrass (consola)

Utilice el siguiente procedimiento para elegir una función de servicio de Greengrass diferente y asociarla a la suya Cuenta de AWS en la que esté seleccionada Región de AWS actualmente en la consola.

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación, seleccione Configuración.
3. En Rol de servicio de Greengrass, seleccione Elegir un rol diferente.

   Se abre el cuadro de diálogo Actualizar el rol de servicio de Greengrass y muestra los roles de IAM Cuenta de AWS que se definen AWS IoT Greengrass como una entidad de confianza.

4. Elija el rol de servicio de Greengrass que desee asignar.
5. Elija Adjuntar rol.

Desasociar el rol de servicio de Greengrass (consola)

Utilice el siguiente procedimiento para separar el rol de servicio de Greengrass de AWS su cuenta actual. Región de AWS Esto revoca los permisos de acceso AWS IoT Greengrass a AWS los servicios actuales. Región de AWS

> **Important**
>
> La desasociación del rol de servicio podría interrumpir las operaciones activas.

1. Vaya a la consola de AWS IoT.
2. En el panel de navegación, seleccione Configuración.
3. En Rol de servicio de Greengras, seleccione Desasociar rol.
4. En el cuadro de diálogo de confirmación, elija Desconectar.

> **Note**
>
> Si ya no necesita el rol, puede eliminarlo en la consola de IAM. Para obtener más información, consulte Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM.
> Es posible que otras funciones te permitan acceder AWS IoT Greengrass a tus recursos. Para buscar todos los roles que permiten que AWS IoT Greengrass asuma los permisos en su nombre, en la consola de IAM, en la página Roles, busque los roles que incluyan AWS service: greengrass en la columna Entidades de confianza.

Administración del rol de servicio de Greengrass (CLI)

En los siguientes procedimientos, asumimos que AWS Command Line Interface está instalado y configurado para usar su Cuenta de AWS. Para obtener más información, consulte Instalar, actualizar y desinstalar la AWS CLI y Configurar la AWS CLI en la Guía del usuario de AWS Command Line Interface .

Puede usarlo AWS CLI para las siguientes tareas de administración de roles:

Temas

- Obtener el rol de servicio de Greengrass (CLI)
- Creación del rol de servicio de Greengrass (CLI)
- Eliminar el rol de servicio de Greengrass (CLI)

Obtener el rol de servicio de Greengrass (CLI)

Utilice el procedimiento siguiente para descubrir si un rol de servicio de Greengrass está asociado a su Cuenta de AWS en una Región de AWS.

- Obtenga el rol de servicio. *region*Reemplácelo por su Región de AWS (por ejemplo,`us-west-2`).

  ```
  aws greengrassv2 get-service-role-for-account --region region
  ```

  Si ya hay un rol de servicio de Greengrass asociado a su cuenta, la solicitud devuelve los siguientes metadatos de rol.

  ```
  {
    "associatedAt": "timestamp",
    "roleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Si la solicitud no devuelve ningún metadato de rol, entonces debe crear el rol de servicio (si no existe) y asociarlo a su cuenta en la Región de AWS.

Creación del rol de servicio de Greengrass (CLI)

Siga los pasos que se indican a continuación para crear un rol y asociarlo a su Cuenta de AWS.

Para crear el rol de servicio mediante IAM

1. Cree un rol con una política de confianza que AWS IoT Greengrass permita asumir el rol. Este ejemplo crea un rol denominado `Greengrass_ServiceRole`, pero puede utilizar un nombre distinto. Le recomendamos que incluya también las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en su política de confianza para ayudar a

prevenir el problema de seguridad del suplente confuso. Las claves de contexto de condición restringen el acceso para permitir solo las solicitudes que provienen de la cuenta especificada y del espacio de trabajo de Greengrass. Para obtener más información sobre el problema del suplente confuso, consulte Prevención de la sustitución confusa entre servicios.

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

2. Copie el ARN del rol de los metadatos del rol en la salida. Puede utilizar el ARN para asociar el rol a su cuenta.
3. Asocie la política de `AWSGreengrassResourceAccessRolePolicy` al rol.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

Para asociar el rol de servicio a su Cuenta de AWS

- Asocie el rol a su cuenta. *`role-arn`*Sustitúyalo por el ARN del rol de servicio y *`region`* por el tuyo Región de AWS (por ejemplo,`us-west-2`).

  ```
  aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
  region region
  ```

  Si se realiza correctamente, la solicitud devuelve la siguiente respuesta.

  ```
  {
    "associatedAt": "timestamp"
  }
  ```

Eliminar el rol de servicio de Greengrass (CLI)

Utilice los pasos siguientes para desasociar el rol de servicio de Greengrass de su Cuenta de AWS.

- Desasocie el rol de servicio de su cuenta. *region*Sustitúyalo por su Región de AWS (por ejemplo,`us-west-2`).

  ```
  aws greengrassv2 disassociate-service-role-from-account --region region
  ```

  Si se ejecuta correctamente, se devuelve la siguiente respuesta.

  ```
  {
    "disassociatedAt": "timestamp"
  }
  ```

  > **Note**
  >
  > Deberías eliminar la función de servicio si no la utilizas en ninguna Región de AWS. Use primero delete-role-policy para desasociar la política administrada `AWSGreengrassResourceAccessRolePolicy` del rol y, a continuación, utilice delete-role para eliminar el rol. Para obtener más información, consulte Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM.

Véase también

- Crear un rol para delegar permisos a un AWS servicio en la Guía del usuario de IAM
- Modificación de un rol en la Guía del usuario de IAM
- Eliminación de roles o perfiles de instancia en la Guía del usuario de IAM
- AWS IoT Greengrass comandos de la Referencia de AWS CLI comandos
  - associate-service-role-to-cuenta
  - disassociate-service-role-from-cuenta
  - get-service-role-for-cuenta
- Comandos de IAM en la Referencia de los comandos de AWS CLI
  - attach-role-policy
  - create-role

- delete-role
- delete-role-policy

AWS políticas gestionadas para AWS IoT Greengrass

Una política AWS administrada es una política independiente creada y administrada por AWS. AWS Las políticas administradas están diseñadas para proporcionar permisos para muchos casos de uso comunes, de modo que pueda empezar a asignar permisos a usuarios, grupos y funciones.

Ten en cuenta que es posible que las políticas AWS administradas no otorguen permisos con privilegios mínimos para tus casos de uso específicos, ya que están disponibles para que los usen todos los AWS clientes. Se recomienda definir políticas administradas por el cliente específicas para sus casos de uso a fin de reducir aún más los permisos.

No puedes cambiar los permisos definidos en AWS las políticas administradas. Si AWS actualiza los permisos definidos en una política AWS administrada, la actualización afecta a todas las identidades principales (usuarios, grupos y roles) a las que está asociada la política. AWS es más probable que actualice una política AWS administrada cuando Servicio de AWS se lance una nueva o cuando estén disponibles nuevas operaciones de API para los servicios existentes.

Para obtener más información, consulte Políticas administradas por AWS en la Guía del usuario de IAM.

Temas

- AWS política gestionada: AWSGreengrass FullAccess
- AWS política gestionada: AWSGreengrass ReadOnlyAccess
- AWS política gestionada: AWSGreengrass ResourceAccessRolePolicy
- AWS IoT Greengrass actualizaciones de las políticas AWS gestionadas

AWS política gestionada: AWSGreengrass FullAccess

Puede asociar la política `AWSGreengrassFullAccess` a las identidades de IAM.

Esta política otorga permisos administrativos que brindan a una entidad principal acceso completo a todas las acciones de AWS IoT Greengrass .

Detalles de los permisos

Esta política incluye los permisos siguientes:

- `greengrass`: permite a las entidades principales obtener acceso completo a todas las acciones de AWS IoT Greengrass .

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS política gestionada: AWSGreengrass ReadOnlyAccess

Puede asociar la política `AWSGreengrassReadOnlyAccess` a las identidades de IAM.

Esta política concede permisos de solo lectura que permiten a una entidad principal visualizar, pero no modificar, la información de AWS IoT Greengrass. Por ejemplo, las entidades principales con estos permisos pueden ver la lista de componentes implementados en un dispositivo principal de Greengrass, pero no pueden crear una implementación para cambiar los componentes que se ejecutan en ese dispositivo.

Detalles de los permisos

Esta política incluye los permisos siguientes:

- `greengrass`: permite a las entidades principales realizar acciones que devuelven una lista de elementos o detalles sobre un elemento. Esto incluye las operaciones de la API que comienzan con `List` o `Get`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:List*",
                "greengrass:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS política gestionada: AWSGreengrass ResourceAccessRolePolicy

Puede adjuntar la `AWSGreengrassResourceAccessRolePolicy` política a sus entidades de IAM. AWS IoT Greengrass también vincula esta política a un rol de servicio que le permite AWS IoT Greengrass realizar acciones en su nombre. Para obtener más información, consulte Rol de servicio de Greengrass.

Esta política otorga permisos administrativos que permiten AWS IoT Greengrass realizar tareas esenciales, como recuperar las funciones de Lambda, AWS IoT administrar las sombras de los dispositivos y verificar los dispositivos cliente de Greengrass.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `greengrass`: administrar los recursos de Greengrass.
- `iot(*Shadow)` — Gestione AWS IoT las sombras que tengan los siguientes identificadores especiales en sus nombres. Estos permisos son necesarios para que AWS IoT Greengrass pueda comunicarse con los dispositivos principales.
  - `*-gci`— AWS IoT Greengrass utiliza esta sombra para almacenar la información de conectividad de los dispositivos principales, de modo que los dispositivos cliente puedan detectar los dispositivos principales y conectarse a ellos.

  - `*-gcm`— AWS IoT Greengrass V1 usa esta sombra para notificar al dispositivo principal que el certificado de la autoridad de certificación (CA) del grupo Greengrass ha rotado.
  - `*-gda`— AWS IoT Greengrass V1 usa esta sombra para notificar al dispositivo principal acerca de una implementación.
  - `GG_*`: no se usa.
- `iot(DescribeThingyDescribeCertificate)` — Recupera información sobre AWS IoT cosas y certificados. Estos permisos son necesarios para AWS IoT Greengrass poder verificar los dispositivos cliente que se conectan a un dispositivo principal. Para obtener más información, consulte Interacción con dispositivos IoT locales.
- `lambda`— Recuperar información sobre AWS Lambda las funciones. Este permiso es necesario para que AWS IoT Greengrass V1 pueda implementar funciones Lambda en los núcleos de Greengrass. Para obtener más información, consulte Ejecutar la función Lambda en el AWS IoT Greengrass núcleo de la Guía para desarrolladores de la AWS IoT Greengrass V1.
- `secretsmanager`— Recupera el valor de AWS Secrets Manager los secretos cuyos nombres comiencen por. `greengrass-` Este permiso es necesario para que la AWS IoT Greengrass versión 1 pueda implementar los secretos de Secrets Manager en los núcleos de Greengrass. Para obtener más información, consulte Implementar secretos en el AWS IoT Greengrass núcleo en la Guía para desarrolladores de AWS IoT Greengrass la versión 1.
- `s3`: recupera archivos y objetos de buckets de S3 cuyos nombres contengan `greengrass` o `sagemaker`. Estos permisos son necesarios para que la AWS IoT Greengrass versión 1 pueda implementar los recursos de aprendizaje automático que almacene en los depósitos de S3. Para obtener más información, consulte los recursos de aprendizaje automático en la Guía para desarrolladores de AWS IoT Greengrass la versión 1.
- `sagemaker`— Recuperar información sobre los modelos de inferencia de aprendizaje automático de Amazon SageMaker AI. Este permiso es necesario para que la AWS IoT Greengrass versión 1 pueda implementar modelos de aprendizaje automático en los núcleos de Greengrass. Para obtener más información, consulte Realizar inferencias de aprendizaje automático en la Guía para desarrolladores de la AWS IoT Greengrass versión 1.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "AllowGreengrassAccessToShadows",
            "Action": [
                "iot:DeleteThingShadow",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
```

```
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
```

```
    ]
}
```

AWS IoT Greengrass actualizaciones de las políticas AWS gestionadas

Puede ver los detalles sobre las actualizaciones de las políticas AWS administradas AWS IoT Greengrass desde el momento en que este servicio comenzó a rastrear estos cambios. Para recibir alertas automáticas sobre los cambios en esta página, suscríbase a la fuente RSS de la página de historial AWS IoT Greengrass de documentos de la versión 2.

Cambio	Descripción	Fecha
AWS IoT Greengrass comenzó a rastrear los cambios	AWS IoT Greengrass comenzó a realizar un seguimiento de los cambios de sus políticas AWS gestionadas.	2 de julio de 2021

Prevención de la sustitución confusa entre servicios

El problema de la sustitución confusa es un problema de seguridad en el que una entidad que no tiene permiso para realizar una acción puede obligar a una entidad con más privilegios a realizar la acción. En este caso AWS, la suplantación de identidad entre varios servicios puede provocar el confuso problema de un diputado. La suplantación entre servicios puede producirse cuando un servicio (el servicio que lleva a cabo las llamadas) llama a otro servicio (el servicio al que se llama). El servicio que lleva a cabo las llamadas se puedes manipular para utilizar sus permisos a fin de actuar en función de los recursos de otro cliente de una manera en la que no debe tener permiso para acceder. Para evitarlo, AWS proporciona herramientas que lo ayudan a proteger sus datos para todos los servicios con entidades principales de servicio a las que se les ha dado acceso a los recursos de su cuenta.

Se recomienda utilizar las claves de contexto de condición `aws:SourceAccount`global `aws:SourceArn`y las claves de contexto en las políticas de recursos para limitar los permisos que se AWS IoT Greengrass otorgan a otro servicio al recurso. Si se utilizan ambas claves contextuales de condición global, el valor `aws:SourceAccount` y la cuenta del valor `aws:SourceArn` deben utilizar el mismo ID de cuenta cuando se utilicen en la misma declaración de política.

El valor de `aws:SourceArn` debe ser el recurso de cliente de Greengrass asociado a la solicitud `sts:AssumeRole`.

La forma más eficaz de protegerse contra el problema de la sustitución confusa es utilizar la clave de contexto de condición global de `aws:SourceArn` con el ARN completo del recurso. Si no conoce el ARN completo del recurso o si especifica varios recursos, utiliza la clave de condición de contexto global `aws:SourceArn` con comodines (*) para las partes desconocidas del ARN. Por ejemplo, `arn:aws:greengrass::account-id:*`.

Para ver un ejemplo de una política que usa las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount`, consulte Creación del rol de servicio de Greengrass.

Solución de problemas de identidad y acceso para AWS IoT Greengrass

Utilice la siguiente información como ayuda para diagnosticar y solucionar los problemas más comunes que pueden surgir al trabajar con un AWS IoT Greengrass IAM.

Problemas

- No estoy autorizado a realizar ninguna acción en AWS IoT Greengrass
- No estoy autorizado a realizar lo siguiente: PassRole
- Soy administrador y quiero permitir el acceso de otras personas AWS IoT Greengrass
- Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS IoT Greengrass recursos

Para obtener ayuda general de solución de problemas, consulte Resolución de problemas.

No estoy autorizado a realizar ninguna acción en AWS IoT Greengrass

Si recibe un error que indica que no está autorizado para llevar a cabo una acción, debe ponerse en contacto con su administrador para recibir ayuda. Su administrador es la persona que le facilitó su nombre de usuario y contraseña.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM `mateojackson` intenta ver detalles sobre un dispositivo núcleo, pero no tiene permisos `greengrass:GetCoreDevice`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

En este caso, Mateo pide a su administrador que actualice sus políticas de forma que pueda obtener acceso al recurso `arn:aws:greengrass:us-west-2:123456789012:coreDevices/MyGreengrassCore` mediante la acción `greengrass:GetCoreDevice`.

A continuación se indican problemas de IAM generales que puede encontrar al trabajar con AWS IoT Greengrass.

No estoy autorizado a realizar lo siguiente: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, las políticas deben actualizarse a fin de permitirle pasar un rol a AWS IoT Greengrass.

Algunas Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir la función al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en AWS IoT Greengrass. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su AWS administrador. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Soy administrador y quiero permitir el acceso de otras personas AWS IoT Greengrass

Para permitir el acceso de otras personas AWS IoT Greengrass, debes conceder permiso a las personas o aplicaciones que necesitan acceso. Si usa AWS IAM Identity Center para administrar las personas y las aplicaciones, debe asignar conjuntos de permisos a los usuarios o grupos para definir su nivel de acceso. Los conjuntos de permisos crean políticas de IAM y las asignan a los roles de IAM asociados a la persona o aplicación de forma automática. Para obtener más información, consulte la sección Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .

Si no utiliza IAM Identity Center, debe crear entidades de IAM (usuarios o roles) para las personas o aplicaciones que necesitan acceso. A continuación, debe asociar una política a la entidad que

le conceda los permisos correctos en AWS IoT Greengrass. Una vez concedidos los permisos, proporcione las credenciales al usuario o al desarrollador de la aplicación. Utilizarán esas credenciales para acceder a AWS. Para obtener más información sobre la creación de usuarios, grupos, políticas y permisos de IAM, consulte Identidades de IAM y Políticas y permisos en IAM en la Guía del usuario de IAM.

Quiero permitir que personas ajenas a mí accedan Cuenta de AWS a mis AWS IoT Greengrass recursos

Puede crear una función de IAM que los usuarios de otras cuentas o personas ajenas a su organización puedan utilizar para acceder a sus AWS recursos. Puede especificar una persona de confianza para que asuma el rol. Para obtener más información, consulte Proporcionar acceso a un usuario de IAM en otro Cuenta de AWS que sea de su propiedad y Proporcionar acceso a Cuenta de AWS un usuario de IAM que sea propiedad de terceros en la Guía del usuario de IAM.

AWS IoT Greengrass no admite el acceso entre cuentas en función de políticas basadas en recursos o listas de control de acceso (). ACLs

Cómo permitir el tráfico del dispositivo a través de un proxy o firewall

Los dispositivos principales y los componentes de Greengrass realizan solicitudes salientes a AWS servicios y otros sitios web. Como medida de seguridad, puede limitar el tráfico saliente a una pequeña variedad de puntos de conexión y puertos. Puede usar la siguiente información sobre los puntos de conexión y los puertos para limitar el tráfico de los dispositivos a través de un proxy, un firewall o un grupo de seguridad de Amazon VPC. Para obtener más información acerca de cómo configurar un dispositivo principal para usar un proxy, consulte Realizar la conexión en el puerto 443 o a través de un proxy de red.

Temas

- Puntos de conexión para el funcionamiento básico
- Puntos de conexión para la instalación con aprovisionamiento automático
- Puntos finales para los componentes AWS proporcionados

Puntos de conexión para el funcionamiento básico

Los dispositivos principales de Greengrass usan los siguientes puntos de conexión y puertos para su funcionamiento básico.

AWS IoT Recupere los puntos finales

Obtenga los AWS IoT puntos finales que desee y guárdelos para usarlos más adelante. Cuenta de AWS El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Haga lo siguiente:

1. Obtenga el punto final AWS IoT de datos para su. Cuenta de AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenga el punto final de AWS IoT credenciales para su Cuenta de AWS.

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`greengrass-ats.iot. region.amazonaws.com`	8443 o 443	Sí	Se usa para las operacion

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			es del plano de datos, como la instalación de implementaciones y el trabajo con dispositivos de cliente.
device-data-prefix `-ats.iot.` *region* `.amazonaws.com`	MQTT: 8883 o 443 HTTPS: 8443 o 443	Sí	Se usa para las operaciones del plano de datos en la administración de dispositivos, como la comunicación MQTT y la sincronización de sombras con AWS IoT Core.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	Sí	Se utiliza para adquirir AWS credenciales, que el dispositivo principal utiliza para descargar artefactos de componentes de Amazon S3 y realizar otras operaciones. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			AWS los servicios.

`*.s3.amazonaws.com`
 	
`*.s3.region.amazonaws.com`
	443	Sí	Se usa para las implementaciones. Este formato incluye el carácter *, porque los prefijos del punto de conexión se controlan internamente y pueden cambiar en cualquier momento.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`data.iot.` *`region`*`.amazonaws.com`	443	No	Es obligatorio si el dispositivo principal ejecuta una versión del núcleo de Greengrass anterior a la versión 2.4.0 y está configurado para usar un proxy de red. El dispositivo principal utiliza este punto final para la comunicación MQTT AWS IoT Core cuando se encuentra detrás de un

punto de enlace	Puerto	Obligatorio	Description (Descripción)
			proxy. Para obtener más información, consulte Configuración de un proxy de red.

Puntos de conexión para la instalación con aprovisionamiento automático

Los dispositivos principales de Greengrass utilizan los siguientes puntos finales y puertos al instalar el software AWS IoT Greengrass Core con aprovisionamiento automático de recursos.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`iot.region.amazonaws.com`	443	Sí	Se utiliza para crear AWS IoT recursos y recuperar información sobre los recursos existentes. AWS IoT

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`iam.amazonaws.com`	443	Sí	Se usa para crear recursos de IAM y recuperar información sobre los recursos de IAM existentes.
`sts.region.amazonaws.com`	443	Sí	Se utiliza para obtener el ID de su Cuenta de AWS.

punto de enlace	Puerto	Obligatorio	Description (Descripción)
`greengrass. region.amazonaws.com`	443	No	Es obligatorio si usa el argumento `--deploy-dev-tools` para implementar el componente de la CLI de Greengrass en el dispositivo principal.

Puntos finales para los componentes AWS proporcionados

Los dispositivos principales de Greengrass usan puntos de conexión adicionales en función de los componentes de software que ejecuten. Puedes encontrar los puntos finales que requiere cada componente AWS proporcionado en la sección de requisitos de la página de cada componente de esta guía para desarrolladores. Para obtener más información, consulte Componentes proporcionados por AWS.

Validación de conformidad en AWS IoT Greengrass

Para saber si un Servicio de AWS está incluido en el ámbito de programas de conformidad específicos, consulte Servicios de AWS incluidos por programa de conformidad y escoja el

programa de conformidad que le interese. Para obtener información general, consulte Programas de conformidad de AWS.

Puedes descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la confidencialidad de los datos, los objetivos de conformidad de su empresa, así como de la legislación y los reglamentos aplicables. Para obtener más información sobre la responsabilidad de conformidad al usar Servicios de AWS, consulte la Documentación de seguridad de AWS.

Puntos de conexión de FIPS

AWS IoT Greengrass admite el uso de terminales FIPS (estándar federal de procesamiento de información (FIPS) 140-2). Cuando el modo FIPS está habilitado, todos los flujos de datos, incluidos los protocolos HTTP y MQTT, a los servicios de Nube de AWS deben invocar y establecer conexiones con los puntos de conexión compatibles con FIPS correspondientes (FIPS: Amazon Web Services (AWS)).

Las comunicaciones MQTT AWS IoT utilizarán el punto final FIPS del plano de datos de IoT (Conectarse a puntos finales AWS IoT FIPS - AWS IoT Core) y la biblioteca criptográfica AWS aws-lc, desarrollada y compatible con FIPS.

Para las comunicaciones HTTP en Greengrass:

- En el caso de los componentes core y plugin, todos los clientes HTTP del SDK se configuran con puntos de enlace FIPS; para ello, se establece la propiedad del sistema en true; AWS_USE_FIPS_ENDPOINT
- En el caso de los componentes genéricos, todos los componentes comienzan con la propiedad del sistema AWS_USE_FIPS_ENDPOINT establecida en true. Este proceso garantiza que los clientes HTTP de SDK usados por estos componentes genéricos envíen solicitudes a los puntos de conexión compatibles con FIPS.

> **Note**
>
> En el caso del administrador de flujos, Nucleus pasa la variable de entorno AWS_GG _FIPS_MODE. Esta variable de entorno permite a los clientes HTTP usados en el

administrador de flujos identificar y conectarse al punto de conexión compatible con FIPS correspondiente.

AWS IoT Greengrass ofrece dos métodos para habilitar el modo FIPS: aprovisionamiento e implementación. Para activar el modo FIPS, debe establecer el parámetro de configuración en true. `fipsMode` A continuación, Nucleus establece la propiedad AWS_USE_FIPS_ENDPOINT del sistema en true y la propaga como una variable de entorno a todos los demás componentes. Además, AWS IoT Greengrass descargará un certificado de CA raíz (CA3) y lo anexará al archivo RootCA.pem (o .pem) existente. AmazonRoot CA1 Si habilita FIPS mediante una nueva implementación, el núcleo se reiniciará para garantizar que la propiedad del sistema surta efecto después de habilitar el modo FIPS.

Además de configurar el parámetro `fipsMode`, también debe configurar los parámetros `iotDataEndpoint`, `iotCredEndpoint` y `greengrassDataEndpoint`. Para obtener más información, consulte la documentación relevante a continuación.

Habilitación de los puntos de conexión de FIPS con la implementación

Obtenga sus AWS IoT Cuenta de AWS puntos de conexión y guárdelos para usarlos más adelante. El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Se requieren dos puntos de conexión, el `iotDataEndpoint` y el `iotCredEndpoint`. Haga lo siguiente:

1. Obtenga el punto de conexión de datos de FIPS para su región en los puntos de conexión del plano de datos de FIPS de AWS IoT Core. El punto final de datos FIPS para usted Cuenta de AWS debería tener este aspecto: *`data.iot-fips.us-west-2.amazonaws.com`*
2. Obtenga el punto de conexión de las credenciales de FIPS para su región en los puntos de conexión del plano de datos de FIPS de AWS IoT Core. El punto final de sus credenciales FIPS Cuenta de AWS debería tener este aspecto: *`data.credentials.iot-fips.us-west-2.amazonaws.com`*

A continuación, para habilitar el FIPS con una implementación, debe aplicar la siguiente configuración al núcleo. La configuración que se va a combinar en la implementación es la siguiente.

Console

Configuración de combinación

```
{
  "fipsMode": "true",
  "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
  "greengrassDataPlaneEndpoint": "iotData",
  "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"
}
```

AWS CLI

El siguiente comando crea una implementación a un dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

El archivo `dashboard-deployment.json` contiene el siguiente documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "aws.greengrass.Nucleus": {
      "componentVersion": "2.13.0",
      "configurationUpdate": {
        "merge":{\"fipsMode\":\"true\",\"iotDataEndpoint\":\"data.iot-fips.us-
west-2.amazonaws.com\",\"greengrassDataPlaneEndpoint\":\"iotData\",\"iotCredEndpoint
\":\"data.credentials.iot-fips.us-west-2.amazonaws.com\"}"
      }
    }
  }
}
```

Greengrass CLI

El siguiente comando de la CLI de Greengrass crea una implementación local en un dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
```

```
  --artifactDir artifacts \
  --merge "aws.greengrass.Nucleus=2.13.0" \
  --update-config dashboard-configuration.json
```

El archivo `dashboard-configuration.json` contiene el siguiente documento JSON.

```
{
  "aws.greengrass.Nucleus": {
    "MERGE": {
       "fipsMode": "true",
       "iotDataEndpoint": "data.iot-fips.us-west-2.amazonaws.com",
       "greengrassDataPlaneEndpoint": "iotData",
       "iotCredEndpoint": "data.credentials.iot-fips.us-west-2.amazonaws.com"

    }
  }
}
```

Instale el núcleo con puntos de conexión de FIPS con aprovisionamiento manual de recursos

Aprovisione manualmente AWS los recursos para los dispositivos AWS IoT Greengrass V2 principales con puntos finales FIPS

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Recupere los puntos finales AWS IoT
- Crea cualquier cosa AWS IoT
- Creación del certificado del objeto
- Creación de un rol de intercambio de token
- Descarga de certificados al dispositivo

- Configuración del entorno del dispositivo
- Descargue el software AWS IoT Greengrass principal
- Instale el software principal AWS IoT Greengrass

Recupere los puntos finales AWS IoT

Obtenga sus AWS IoT Cuenta de AWS puntos finales y guárdelos para usarlos más tarde. El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Se requieren dos puntos de conexión, el `iotDataEndpoint` y el `iotCredEndpoint`. Haga lo siguiente:

1. Obtenga el punto de conexión de datos de FIPS para su región en los puntos de conexión del plano de datos de FIPS de AWS IoT Core. El punto final de datos FIPS para usted Cuenta de AWS debería tener este aspecto: *`data.iot-fips.us-west-2.amazonaws.com`*
2. Obtenga el punto de conexión de las credenciales de FIPS para su región en los puntos de conexión del plano de datos de FIPS de AWS IoT Core. El punto final de sus credenciales FIPS Cuenta de AWS debería tener este aspecto: *`data.credentials.iot-fips.us-west-2.amazonaws.com`*

Crea cualquier cosa AWS IoT

AWS IoT las cosas representan dispositivos y entidades lógicas a las que se conectan AWS IoT. Los dispositivos principales de Greengrass son AWS IoT cosas. Cuando registras un dispositivo como una AWS IoT cosa, ese dispositivo puede usar un certificado digital para autenticarse. AWS

En esta sección, crearás AWS IoT algo que represente tu dispositivo.

Para crear cualquier AWS IoT cosa

1. Crea cualquier AWS IoT cosa para tu dispositivo. En su equipo de desarrollo, ejecute el siguiente comando.

   - *`MyGreengrassCore`*Sustitúyalo por el nombre de la cosa a utilizar. Este nombre también es el nombre de su dispositivo principal de Greengrass.

     > **Note**
     >
     > El nombre del objeto no puede contener dos puntos (`:`).

```
aws iot create-thing --thing-name MyGreengrassCore
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Opcional) Añada la AWS IoT cosa a un grupo de cosas nuevo o existente. Los grupos de objetos se usan para administrar las flotas de dispositivos principales de Greengrass. Al implementar componentes de software en sus dispositivos, puede dirigirlos a dispositivos individuales o a grupos de dispositivos. Puede agregar un dispositivo a un grupo de objetos con una implementación activa de Greengrass para implementar los componentes de software de ese grupo de objetos en el dispositivo. Haga lo siguiente:

   a. (Opcional) Cree un grupo de AWS IoT cosas.

      - *MyGreengrassCoreGroup*Sustitúyalo por el nombre del grupo de cosas que desee crear.

        > **Note**
        >
        > El nombre del grupo de objetos no puede contener dos puntos (`:`).

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

b. Añada la AWS IoT cosa a un grupo de cosas.

- *MyGreengrassCore*Sustitúyala por el nombre de la AWS IoT cosa.
- *MyGreengrassCoreGroup*Sustitúyalo por el nombre del grupo de cosas.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Creación del certificado del objeto

Al registrar un dispositivo como una AWS IoT cosa, ese dispositivo puede utilizar un certificado digital para autenticarse AWS. Este certificado permite que el dispositivo se comunique con AWS IoT y AWS IoT Greengrass.

En esta sección, puede crear y descargar certificados que el dispositivo puede usar para conectarse a AWS.

Si desea configurar el software AWS IoT Greengrass principal para que utilice un módulo de seguridad de hardware (HSM) para almacenar de forma segura la clave privada y el certificado, siga los pasos para crear el certificado a partir de una clave privada de un HSM. De lo contrario, siga los pasos para crear el certificado y la clave privada en el AWS IoT servicio. La característica de seguridad de hardware solo está disponible en dispositivos Linux. Para obtener más información sobre la seguridad del hardware y los requisitos para su uso, consulte Integración de la seguridad de hardware.

Cree el certificado y la clave privada en el AWS IoT servicio

Creación del certificado del objeto

1. Crea una carpeta donde descargues los certificados de la AWS IoT cosa.

```
mkdir greengrass-v2-certs
```

2. Crea y descarga los certificados de la AWS IoT cosa.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
```

```
"
  }
}
```

Guarde el nombre de recurso de Amazon (ARN) del certificado para usarlo para configurar el certificado más adelante.

Creación del certificado a partir de una clave privada en un HSM

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Creación del certificado del objeto

1. En el dispositivo principal, inicialice un token PKCS#11 en el HSM y genere una clave privada. La clave privada debe ser una clave RSA con un tamaño de clave RSA-2048 (o mayor) o una clave ECC.

   > **Note**
   >
   > Para utilizar un módulo de seguridad de hardware con claves ECC, debe utilizar la versión del núcleo de Greengrass 2.5.6 o posterior.
   > Para usar un módulo de seguridad de hardware y el administrador de secretos, debe usar un módulo de seguridad de hardware con claves RSA.

   Consulte la documentación de su HSM para obtener información sobre cómo inicializar el token y generar la clave privada. Si su HSM admite objetos IDs, especifique un ID de objeto al generar la clave privada. Guarde el ID de ranura, el PIN de usuario, la etiqueta del objeto y el ID del objeto (si su HSM utiliza alguno) que especifique al inicializar el token y generar la clave privada. Estos valores se utilizan más adelante cuando se importa el certificado de la cosa al HSM y se configura el software AWS IoT Greengrass principal.
2. Cree una solicitud de firma de certificado (CSR) de la clave privada. AWS IoT usa esta CSR para crear un certificado de objetos para la clave privada que usted generó en el HSM. Para

obtener información sobre cómo crear una CSR de la clave privada, consulte la documentación de su HSM. La CSR es un archivo, como `iotdevicekey.csr`.

3. Copie la CSR del dispositivo a su computadora de desarrollo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir la CSR. *`device-ip-address`*Sustitúyalo por la dirección IP del dispositivo y *`~/iotdevicekey.csr`* sustitúyalo por la ruta al archivo CSR del dispositivo.

   ```
   scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
   ```

4. En tu ordenador de desarrollo, crea una carpeta en la que descargues el certificado del dispositivo AWS IoT .

   ```
   mkdir greengrass-v2-certs
   ```

5. Use el archivo CSR para crear y descargar el certificado para el objeto AWS IoT en su computadora de desarrollo.

   ```
   aws iot create-certificate-from-csr --set-as-active --certificate-signing-
   request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
   device.pem.crt
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificateId":
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificatePem": "-----BEGIN CERTIFICATE-----
   MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
    0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
    WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
    EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
    jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
    MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
    WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
    HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
    BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
    k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
   ```

```
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Guarde el ARN del certificado para usarlo más adelante para configurarlo.

A continuación, configura el certificado de la cosa. Para obtener más información, consulte Configuración del certificado del objeto.

Creación de un rol de intercambio de token

Los dispositivos principales de Greengrass utilizan una función de servicio de IAM, denominada función de intercambio de fichas, para autorizar las llamadas a los servicios. AWS El dispositivo utiliza el proveedor de AWS IoT credenciales para obtener AWS credenciales temporales para esta función, lo que permite al dispositivo interactuar con Amazon Logs AWS IoT, enviar registros a Amazon CloudWatch Logs y descargar artefactos de componentes personalizados de Amazon S3. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Se utiliza un alias de AWS IoT rol para configurar el rol de intercambio de fichas para los dispositivos principales de Greengrass. Los alias de rol le permiten cambiar el rol de intercambio de token de un dispositivo, pero mantener la configuración del dispositivo igual. Para obtener más información, consulte Autorización de llamadas a los servicios de AWS en la Guía para desarrolladores de AWS IoT Core .

En esta sección, creará un rol de IAM de intercambio de tokens y un alias de AWS IoT rol que apunte al rol. Si ya ha configurado un dispositivo principal de Greengrass, puede usar su rol de intercambio de token y su alias de rol en lugar de crear otros nuevos. A continuación, configure el objeto AWS IoT del dispositivo para que use ese rol y ese alias.

Creación de un rol de IAM de intercambio de token

1. Creación de un rol de IAM que su dispositivo puede usar como rol de intercambio de token. Haga lo siguiente:

a. Creación de un archivo que contenga el documento de política de confianza que requiere el rol de intercambio de token.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano device-role-trust-policy.json
```

   Copie el siguiente JSON en el archivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Creación del rol de intercambio de token con el documento de política de confianza.

   - *GreengrassV2TokenExchangeRole*Sustitúyalo por el nombre del rol de IAM que se va a crear.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
```

```
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Creación de un archivo que contenga el documento de política de acceso que requiere el rol de intercambio de token.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

```
nano device-role-access-policy.json
```

   Copie el siguiente JSON en el archivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Esta política de acceso no permite el acceso a los artefactos de componentes en los buckets de S3. Para implementar componentes personalizados que definan artefactos en Amazon S3, debe agregar permisos al rol para permitir que su dispositivo principal recupere artefactos de componentes. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.
> Si aún no tiene un bucket de S3 para los artefactos de los componentes, puede agregar estos permisos más adelante, después de crear un bucket.

d. Creación de la política de IAM a partir del documento de política.

- *GreengrassV2TokenExchangeRoleAccess*Sustitúyalo por el nombre de la política de IAM que se va a crear.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Adjunte la política de IAM al rol de intercambio de token.

- Reemplace *GreengrassV2TokenExchangeRole* por el nombre del rol de IAM.
- Reemplace el ARN de la política por el ARN de la política de IAM que creó en el paso anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

2. Cree un alias de AWS IoT rol que apunte al rol de intercambio de fichas.

- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias del rol que se va a crear.
- Reemplace el ARN del rol por el ARN del rol de IAM que creó en el paso anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para crear un alias de rol, debe tener el permiso para transferir el rol de IAM de intercambio de token a AWS IoT. Si recibe un mensaje de error al intentar crear un alias de rol, compruebe que el AWS usuario tiene este permiso. Para obtener más información, consulte Conceder permisos a un usuario para transferir un rol a un AWS servicio en la Guía del AWS Identity and Access Management usuario.

3. Cree y adjunte una AWS IoT política que permita a su dispositivo principal de Greengrass utilizar el alias del rol para asumir el rol de intercambio de fichas. Si ya ha configurado un dispositivo

principal de Greengrass, puede adjuntar su AWS IoT política de alias de rol en lugar de crear una nueva. Haga lo siguiente:

a. (Opcional) Cree un archivo que contenga el documento AWS IoT de política que requiere el alias del rol.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano greengrass-v2-iot-role-alias-policy.json
   ```

   Copie el siguiente JSON en el archivo.

   - Reemplace el ARN del recurso por el ARN del alias de rol.

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "iot:AssumeRoleWithCertificate",
         "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
   GreengrassCoreTokenExchangeRoleAlias"
       }
     ]
   }
   ```

b. Cree una AWS IoT política a partir del documento de política.

   - *GreengrassCoreTokenExchangeRoleAliasPolicy*Sustitúyala por el nombre de la AWS IoT política que se va a crear.

   ```
   aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
    --policy-document file://greengrass-v2-iot-role-alias-policy.json
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
   ```

```
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17       \\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Adjunta la AWS IoT política al certificado de la AWS IoT cosa.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Sustitúyala por el nombre de la AWS IoT política de alias del rol.
- Reemplace el ARN de destino por el ARN del certificado de su objeto AWS IoT .

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

El comando no tiene ningún resultado si la solicitud se realiza correctamente.

Descarga de certificados al dispositivo

Anteriormente, descargó el certificado de su dispositivo en su computadora de desarrollo. En esta sección, copie el certificado en el dispositivo principal para configurar el dispositivo con los certificados que usa para conectarse a AWS IoT. También descargue el certificado de la autoridad del certificado raíz (CA) de Amazon. Si usa un HSM, también importe el archivo de certificado al HSM en esta sección.

- Si anteriormente creó el elemento certificado y clave privada en el AWS IoT servicio, siga los pasos para descargar los certificados con la clave privada y los archivos de certificado.

- Si anteriormente creó el certificado del objeto de una clave privada en un módulo de seguridad de hardware (HSM), siga los pasos para descargar los certificados con la clave privada y el certificado en un HSM.

Descargar certificados con clave privada y archivos de certificado

Descarga de certificados al dispositivo

1. Copia el AWS IoT certificado del objeto desde tu ordenador de desarrollo al dispositivo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir el certificado. *`device-ip-address`*Sustitúyalo por la dirección IP de tu dispositivo.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Cree la carpeta raíz de Greengrass en el dispositivo. Más adelante instalarás el software AWS IoT Greengrass principal en esta carpeta.

   > **Note**
   >
   > Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta que desee utilizar.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   mkdir C:\greengrass\v2
   ```

PowerShell

- Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

```
mkdir C:\greengrass\v2
```

3. (Solo para Linux) Establezca los permisos de la carpeta principal de la carpeta raíz de Greengrass.

   - */greengrass*Sustitúyalo por el elemento principal de la carpeta raíz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Copia los certificados de la AWS IoT cosa a la carpeta raíz de Greengrass.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta raíz de Greengrass.

   ```
   sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
   ```

5. Descargue el certificado de la entidad de certificación (CA) de Amazon. Los certificados de AWS IoT están asociados al certificado de CA raíz de Amazon de forma predeterminada. Descargue el CA1 certificado y el CA3certificado.

Linux or Unix

- Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
greengrass/v2/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Descarga de certificados con la clave privada y el certificado en un HSM

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Descarga de certificados al dispositivo

1. Copia el AWS IoT certificado del dispositivo de desarrollo desde tu ordenador de desarrollo al dispositivo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp` de la computadora de desarrollo para transferir el certificado. *device-ip-address*Sustitúyalo por la dirección IP de tu dispositivo.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Cree la carpeta raíz de Greengrass en el dispositivo. Más adelante instalarás el software AWS IoT Greengrass principal en esta carpeta.

   > **Note**
   >
   > Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta que desee utilizar.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Solo para Linux) Establezca los permisos de la carpeta principal de la carpeta raíz de Greengrass.

   - */greengrass*Sustitúyalo por el elemento principal de la carpeta raíz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Importe el archivo de certificado del objeto `~/greengrass-v2-certs/device.pem.crt`, al HSM. Consulte la documentación de su HSM para saber cómo importar certificados allí. Importe el certificado con el mismo token, ID de ranura, PIN de usuario, etiqueta de objeto e ID de objeto (si su HSM usa alguno) con los que generó la clave privada en el HSM anteriormente.

   > **Note**
   >
   > Si generó la clave privada anteriormente sin un ID de objeto y el certificado tiene un ID de objeto, establezca el ID de objeto de la clave privada con el mismo valor que el certificado. Consulte la documentación de su HSM para obtener información sobre cómo configurar el identificador de objeto para el objeto de clave privada.

5. (Opcional) Elimine el archivo de certificado del objeto para que solo exista en el HSM.

   ```
   rm ~/greengrass-v2-certs/device.pem.crt
   ```

6. Descargue el certificado de la entidad de certificación (CA) de Amazon. Los certificados de AWS IoT están asociados al certificado de CA raíz de Amazon de forma predeterminada. Descargue tanto el certificado CA1 como el CA3certificado.

   Linux or Unix

   - Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
   greengrass/v2/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

   ```
   java -version
   ```

   El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   ```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

      ```
      sudo visudo
      ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

      ```
      root    ALL=(ALL:ALL) ALL
      ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:
   a. Pulse la tecla Windows para abrir el menú de inicio.
   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.
   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.
   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.
   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.
   f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

      ```
      C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
      ```

   g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.
3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.
4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

> **Note**
>
> En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

   i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

   ```
   jar verified.
   ```

   ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

   ```
   jar is unsigned.
   ```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *GreengrassInstaller*Sustitúyalo por la carpeta que desee usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.
> Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Instale el software principal AWS IoT Greengrass

Ejecute el instalador con argumentos que especifiquen las siguientes acciones:

- Realice la instalación desde un archivo de configuración parcial que especifique el uso de los recursos de AWS y certificados que creó anteriormente. El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica la configuración de todos los componentes de Greengrass del dispositivo. El instalador crea un archivo de configuración completo a partir del archivo de configuración parcial que usted proporciona.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.
- Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

- Si creó anteriormente el certificado y la clave privada en el AWS IoT servicio, siga los pasos para instalar el software AWS IoT Greengrass Core con los archivos de clave privada y certificado.
- Si anteriormente creó el certificado Thing a partir de una clave privada en un módulo de seguridad de hardware (HSM), siga los pasos para instalar el software AWS IoT Greengrass Core con la clave privada y el certificado en un HSM.

Instale el software AWS IoT Greengrass principal con la clave privada y los archivos de certificado

Para instalar el software AWS IoT Greengrass Core

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *GreengrassInstaller*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema y los parámetros del núcleo de Greengrass.

   ```
   ---
   ```

```
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      fipsMode: "true"
      iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
      greengrassDataPlaneEndpoint: "iotData"
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

A continuación, proceda del modo siguiente:

- Reemplace cada instancia de */greengrass/v2* por la carpeta raíz de Greengrass.
- *MyGreengrassCore*Sustitúyalo por el nombre de la AWS IoT cosa.
- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- *us-west-2*Sustitúyala por la Región de AWS ubicación en la que creaste los recursos.
- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias de la función de intercambio de fichas.
- *iotDataEndpoint*Sustitúyalo por el punto final de AWS IoT datos.
- Sustituya el *iotCredEndpoint* punto final por el de sus AWS IoT credenciales.

3. Ejecute el instalador y especifique `--init-config` para proporcionar el archivo de configuración.

   - Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.
   - Sustituya cada instancia de por *GreengrassInstaller* la carpeta en la que desempaquetó el instalador.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
   ```

```
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

Note

No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`.

Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

4. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Instale el software AWS IoT Greengrass Core con la clave privada y el certificado en un HSM

> **Note**
>
> Esta función está disponible para la versión 2.5.3 y versiones posteriores del componente núcleo de Greengrass. AWS IoT Greengrass actualmente no admite esta función en los dispositivos principales de Windows.

Para instalar el software AWS IoT Greengrass principal

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *GreengrassInstaller*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Para permitir que el software AWS IoT Greengrass principal utilice la clave privada y el certificado del HSM, instale el componente de proveedor PKCS #11 al instalar el software AWS IoT Greengrass principal. El componente de proveedor PKCS#11 es un complemento que puede configurar durante la instalación. Puede descargar la versión más reciente del componente de proveedor PKCS#11 de la siguiente ubicación:

   - https://d2s8p88vqu9w66.cloudfront. net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-latest.jar

   Descargue el complemento de proveedor PKCS#11 en un archivo denominado `aws.greengrass.crypto.Pkcs11Provider.jar`. Sustitúyala por la carpeta que quieras usar. *`GreengrassInstaller`*

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
   aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
   aws.greengrass.crypto.Pkcs11Provider.jar
   ```

   Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

3. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema, los parámetros del núcleo de Greengrass y los parámetros del proveedor PKCS#11.

   ```
   ---
   system:
     certificateFilePath: "/greengrass/v2/device.pem.crt"
     privateKeyPath: "/greengrass/v2/private.pem.key"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
   ```

```
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.16.1"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      fipsMode: "true"
      iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
      greengrassDataPlaneEndpoint: "iotData"
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
```

A continuación, proceda del modo siguiente:

- Sustituya cada instancia del PKCS #11 URIs por la etiqueta de objeto *iotdevicekey* en la que creó la clave privada e importó el certificado.
- Reemplace cada instancia de */greengrass/v2* por la carpeta raíz de Greengrass.
- *MyGreengrassCore*Sustitúyala por el nombre de la AWS IoT cosa.
- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- *us-west-2*Sustitúyala por la Región de AWS ubicación en la que creaste los recursos.
- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias de la función de intercambio de fichas.
- `iotDataEndpoint`Sustitúyalo por el punto final de AWS IoT datos.
- Reemplace el `iotCredEndpoint` por su punto de conexión de credenciales de AWS IoT .
- Reemplace los parámetros de configuración del componente `aws.greengrass.crypto.Pkcs11Provider` por los valores de la configuración del HSM en el dispositivo principal.

4. Ejecute el instalador y especifique `--init-config` para proporcionar el archivo de configuración.

   - Reemplace */greengrass/v2* por la carpeta raíz de Greengrass.
   - Sustituya cada instancia de *GreengrassInstaller* por la carpeta en la que desempaquetó el instalador.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
```

```
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

> **Note**
>
> No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`. Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

5. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Si instaló el software AWS IoT Greengrass Core como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

Para obtener más información sobre cómo configurar y utilizar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Instalación de puntos de conexión de FIPS con aprovisionamiento de flota

Esta característica está disponible para la versión 2.4.0 y versiones posteriores del componente núcleo de Greengrass.

Instale terminales FIPS en su software AWS IoT Greengrass principal y aprovisione la AWS IoT flota de sus dispositivos principales.

> **Note**
>
> Actualmente, el complemento de aprovisionamiento de flota no admite el almacenamiento de archivos de certificados y claves privadas en un módulo de seguridad de hardware (HSM). Para usar un HSM, instale el software AWS IoT Greengrass Core con aprovisionamiento manual.

Para instalar el software AWS IoT Greengrass Core con el aprovisionamiento de AWS IoT flota, debe configurar los recursos Cuenta de AWS que AWS IoT utiliza para aprovisionar los dispositivos principales de Greengrass. Estos recursos incluyen una plantilla de aprovisionamiento, certificados de reclamación y un rol de IAM para el intercambio de token. Después de crear estos recursos, puede reutilizarlos para aprovisionar varios dispositivos principales de una flota. Para obtener

más información, consulte Configurar el aprovisionamiento de AWS IoT flota para los dispositivos principales de Greengrass.

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla con los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Requisitos previos
- Recupere los puntos finales AWS IoT
- Descarga de certificados al dispositivo
- Configuración del entorno del dispositivo
- Descargue el software AWS IoT Greengrass principal
- Descargue el complemento de aprovisionamiento AWS IoT de flotas
- Instale el software AWS IoT Greengrass principal

Requisitos previos

Para instalar el software AWS IoT Greengrass Core con el aprovisionamiento de AWS IoT flota, primero debe configurar el aprovisionamiento de AWS IoT flota para los dispositivos principales de Greengrass. Tras completar estos pasos una vez, puede utilizar el aprovisionamiento de flotas para instalar el software AWS IoT Greengrass Core en cualquier número de dispositivos.

Recupere los puntos finales AWS IoT

Obtenga los puntos finales FIPS para usted y guárdelos para usarlos más adelante. Cuenta de AWS El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Haga lo siguiente:

1. Obtenga el punto de conexión de datos de FIPS para su región en los puntos de conexión del plano de datos de FIPS de AWS IoT Core. El punto final de datos FIPS correspondiente a usted Cuenta de AWS debería tener este aspecto: `data.iot-fips.us-west-2.amazonaws.com`
2. Obtenga el punto de conexión de las credenciales de FIPS para su región en los puntos de conexión del plano de datos de FIPS de AWS IoT Core. El punto final de sus credenciales

FIPS Cuenta de AWS debería tener este aspecto: *`data.credentials.iot-fips.us-west-2.amazonaws.com`*

Descarga de certificados al dispositivo

El dispositivo utiliza un certificado de reclamación y una clave privada para autenticar su solicitud de aprovisionamiento de AWS recursos y adquirir un certificado de dispositivo X.509. Puede incrustar el certificado de reclamación y la clave privada en el dispositivo durante la fabricación o copiar el certificado y la clave en el dispositivo durante la instalación. En esta sección, debe copiar el certificado de reclamación y la clave privada en el dispositivo. También descargue el certificado de la autoridad del certificado raíz (CA) de Amazon en el dispositivo.

> **Important**
>
> Las claves privadas de la notificación de aprovisionamiento deben estar protegidas en todo momento, también en el dispositivo principal de Greengrass. Te recomendamos que utilices CloudWatch las estadísticas y los registros de Amazon para detectar indicios de uso indebido, como el uso no autorizado del certificado de reclamación para aprovisionar dispositivos. Si detecta un uso incorrecto, deshabilite el certificado de notificación de aprovisionamiento para que no se pueda utilizar para el aprovisionamiento de dispositivos. Para obtener más información, consulte Monitorear AWS IoT en la Guía para desarrolladores de AWS IoT Core .
>
> Para ayudarte a gestionar mejor el número de dispositivos y los dispositivos que se registran automáticamente en el tuyo Cuenta de AWS, puedes especificar un enlace previo al aprovisionamiento al crear una plantilla de aprovisionamiento de flotas. Un enlace de preaprovisionamiento es una AWS Lambda función que valida los parámetros de plantilla que proporcionan los dispositivos durante el registro. Por ejemplo, puede crear un enlace previo al aprovisionamiento que compruebe un ID de un dispositivo con una base de datos para comprobar que el dispositivo tiene permiso de aprovisionamiento. Para obtener más información, consulte los enlaces previos al aprovisionamiento en la Guía para desarrolladores de AWS IoT Core .

Cómo descargar los certificados de reclamación al dispositivo

1. Copie el certificado de reclamación y la clave privada en el dispositivo. Si SSH y SCP están habilitados en la computadora de desarrollo y en el dispositivo, puede utilizar el comando `scp`

de la computadora de desarrollo para transferir el certificado de reclamación y la clave privada. El siguiente comando de ejemplo transfiere estos archivos a una carpeta denominada `claim-certs` en la computadora de desarrollo al dispositivo. *`device-ip-address`*Sustitúyala por la dirección IP de tu dispositivo.

```
scp -r claim-certs/ device-ip-address:~
```

2. Cree la carpeta raíz de Greengrass en el dispositivo. Más adelante instalarás el software AWS IoT Greengrass principal en esta carpeta.

   > **Note**
   >
   > Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

   Linux or Unix

   - Reemplace */greengrass/v2* por la carpeta que desee utilizar.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - Reemplace *C:\greengrass\v2* por la carpeta que desee utilizar.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Solo para Linux) Establezca los permisos de la carpeta principal de la carpeta raíz de Greengrass.

- */greengrass*Sustitúyalo por el elemento principal de la carpeta raíz.

```
sudo chmod 755 /greengrass
```

4. Mueva los certificados de reclamación a la carpeta raíz de Greengrass.

   - Sustituya */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.

   Linux or Unix

   ```
   sudo mv ~/claim-certs /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   move %USERPROFILE%\claim-certs C:\greengrass\v2
   ```

   PowerShell

   ```
   mv -Path ~\claim-certs -Destination C:\greengrass\v2
   ```

5. Descargue tanto el CA1 certificado como el CA3certificado.

   Linux or Unix

   ```
   sudo curl -o - https://www.amazontrust.com/repository/AmazonRootCA3.pem >> /
   greengrass/v2/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como dispositivo AWS IoT Greengrass principal.

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - En Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

   ```
   java -version
   ```

   El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   ```

```
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

      ```
      sudo visudo
      ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

      ```
      root    ALL=(ALL:ALL) ALL
      ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:
   a. Pulse la tecla Windows para abrir el menú de inicio.
   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.
   c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.
   d. Elija Variables de entorno... para abrir la ventana Variables de entorno.
   e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.
   f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

      ```
      C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
      ```

   g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.
3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.
4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

> **Note**
>
> En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

   i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

   ```
   jar verified.
   ```

   ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

   ```
   jar is unsigned.
   ```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *`GreengrassInstaller`*Sustitúyalo por la carpeta que desee usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.
Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Descargue el complemento de aprovisionamiento AWS IoT de flotas

Puedes descargar la última versión del complemento de aprovisionamiento de AWS IoT flotas desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar

Note

Puede descargar una versión específica del complemento de aprovisionamiento de AWS IoT flotas desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar. Para obtener más información sobre cada versión del complemento de aprovisionamiento de flota, consulte Registro de cambios del complemento de aprovisionamiento de flotas AWS IoT.

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

El complemento de aprovisionamiento de flota es de código abierto. Para ver su código fuente, consulta el complemento de aprovisionamiento de AWS IoT flotas en GitHub.

Para descargar el complemento de aprovisionamiento de AWS IoT flotas

- En su dispositivo, descargue el complemento de aprovisionamiento de AWS IoT flotas en un archivo denominado. `aws.greengrass.FleetProvisioningByClaim.jar` *`GreengrassInstaller`*Sustitúyalo por la carpeta que desee usar.

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  PowerShell

  ```
  iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -
  OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

Instale el software AWS IoT Greengrass principal

Ejecute el instalador con argumentos que especifiquen las siguientes acciones:

- Instálelo desde un archivo de configuración parcial que especifique el uso del complemento de aprovisionamiento de flotas para aprovisionar AWS recursos. El software AWS IoT Greengrass Core utiliza un archivo de configuración que especifica la configuración de todos los componentes de Greengrass del dispositivo. El instalador crea un archivo de configuración completo a partir del archivo de configuración parcial que usted proporciona y de AWS los recursos que crea el complemento de aprovisionamiento de flotas.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el

uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.

- Configure el software AWS IoT Greengrass principal como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

Para instalar el software AWS IoT Greengrass Core

1. Compruebe la versión del software AWS IoT Greengrass principal.

   - *`GreengrassInstaller`*Sustitúyala por la ruta a la carpeta que contiene el software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del complemento de aprovisionamiento de flota. Para obtener más información acerca de las opciones que puede especificar, consulte Configurar el complemento de aprovisionamiento de AWS IoT flotas.

Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
    configuration:
      fipsMode: "true"
      greengrassDataPlaneEndpoint: "iotData"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
      iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.16.1"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
```

```
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

A continuación, proceda del modo siguiente:

- *2.16.1*Sustitúyala por la versión del software AWS IoT Greengrass principal.
- Sustituya cada instancia de */greengrass/v2* o *C:\greengrass\v2* por la carpeta raíz de Greengrass.

  > **Note**
  >
  > En los dispositivos Windows, debe especificar los separadores de rutas como barras invertidas dobles (\\), como `C:\\greengrass\\v2`.

- *us-west-2*Sustitúyala por la AWS región en la que creaste la plantilla de aprovisionamiento y otros recursos.
- `iotDataEndpoint`Sustitúyalo por su punto final AWS IoT de datos.
- Reemplace el `iotCredentialEndpoint` por su punto de conexión de credenciales de AWS IoT .
- *GreengrassCoreTokenExchangeRoleAlias*Sustitúyalo por el nombre del alias de la función de intercambio de fichas.
- *GreengrassFleetProvisioningTemplate*Sustitúyalo por el nombre de la plantilla de aprovisionamiento de flota.
- Sustituya `claimCertificatePath` por la ruta al certificado de reclamación del dispositivo.
- Sustituya `claimCertificatePrivateKeyPath` por la ruta a la clave privada del certificado de reclamación del dispositivo.

- Sustituya los parámetros de la plantilla (`templateParameters`) por los valores que se usan para aprovisionar el dispositivo. Este ejemplo hace referencia a la plantilla de ejemplo que define los parámetros `ThingName` y `ThingGroupName`.

3. Ejecute el instalador. Especifique `--trusted-plugin` si desea proporcionar el complemento de aprovisionamiento de flota y especifique `--init-config` si desea proporcionar el archivo de configuración.

   - Reemplace */greengrass/v2* por la carpeta raíz de Greengrass.
   - Sustituya cada instancia de por *GreengrassInstaller* la carpeta en la que desempaquetó el instalador.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

   Windows Command Prompt (CMD)

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
     -jar ./GreengrassInstaller/lib/Greengrass.jar ^
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar ^
     --init-config ./GreengrassInstaller/config.yaml ^
     --component-default-user ggc_user ^
     --setup-system-service true
   ```

   PowerShell

   ```
   java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
     -jar ./GreengrassInstaller/lib/Greengrass.jar `
     --trusted-plugin ./GreengrassInstaller/
   aws.greengrass.FleetProvisioningByClaim.jar `
     --init-config ./GreengrassInstaller/config.yaml `
     --component-default-user ggc_user `
   ```

```
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio del sistema. De lo contrario, el instalador no mostrará ningún mensaje si instala el software correctamente.

> **Note**
>
> No puede usar el argumento `deploy-dev-tools` para implementar herramientas de desarrollo locales cuando ejecuta el instalador sin el argumento `--provision true`. Para obtener información sobre cómo implementar la CLI de Greengrass directamente en su dispositivo, consulte Interfaz de la línea de comandos de Greengrass.

4. Verifique la instalación mediante la consulta de los archivos de la carpeta raíz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

Si la instalación se realizó correctamente, la carpeta raíz contiene varias carpetas, como `config`, `packages` y `logs`.

Si instaló el software AWS IoT Greengrass Core como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

Para obtener más información sobre cómo configurar y utilizar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Instalación de puntos de conexión de FIPS con aprovisionamiento automático de recursos

El software AWS IoT Greengrass Core incluye un instalador que configura su dispositivo como un dispositivo principal de Greengrass. Para configurar un dispositivo rápidamente, el instalador puede proporcionar la AWS IoT AWS IoT cosa, el grupo de cosas, la función de IAM y el alias de la AWS IoT función que el dispositivo principal necesita para funcionar. El instalador también puede implementar las herramientas de desarrollo locales en el dispositivo principal, de modo que usted pueda usar el dispositivo para desarrollar y probar componentes de software personalizados. El instalador necesita credenciales de AWS para aprovisionar estos recursos y crear la implementación.

Si no puede proporcionar AWS las credenciales al dispositivo, puede aprovisionar los AWS recursos que el dispositivo principal necesita para funcionar. También puede implementar las herramientas de desarrollo en un dispositivo principal para usarlas como dispositivo de desarrollo. Esto le permite conceder menos permisos al dispositivo al ejecutar el instalador. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.

> **Important**
>
> Antes de descargar el software AWS IoT Greengrass Core, compruebe que su dispositivo principal cumpla los requisitos para instalar y ejecutar el software AWS IoT Greengrass Core v2.0.

Temas

- Configuración del entorno del dispositivo
- Proporcione AWS las credenciales al dispositivo
- Descargue el software AWS IoT Greengrass principal
- Instale el software principal AWS IoT Greengrass

Configuración del entorno del dispositivo

Siga los pasos de esta sección para configurar un dispositivo Linux o Windows para usarlo como su dispositivo principal de AWS IoT Greengrass .

Configuración de un dispositivo Linux

Para configurar un dispositivo Linux para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior. Los siguientes comandos muestran cómo instalar OpenJDK en su dispositivo.

   - Para distribuciones basadas en Debian o en Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuciones basadas en Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - En Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

- En Amazon Linux 2023:

```
sudo dnf install java-11-amazon-corretto -y
```

Cuando se complete la instalación, ejecute el siguiente comando para comprobar que Java se ejecuta en su dispositivo Linux.

```
java -version
```

El comando imprime la versión de Java que se ejecuta en el dispositivo. Por ejemplo, en una distribución basada en Debian, el resultado podría ser similar al siguiente ejemplo.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Cree el usuario y el grupo predeterminado del sistema que ejecutan los componentes del dispositivo. También puede optar por permitir que el instalador del software AWS IoT Greengrass principal cree este usuario y grupo durante la instalación con el argumento del `--component-default-user` instalador. Para obtener más información, consulte Argumentos del instalador.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Compruebe que el usuario que ejecuta el software AWS IoT Greengrass principal (normalmente`root`) tiene permiso para ejecutar `sudo` con cualquier usuario y grupo.

   a. Ejecute el siguiente comando para abrir el archivo `/etc/sudoers`.

   ```
   sudo visudo
   ```

   b. Compruebe que el permiso del usuario se parezca al siguiente ejemplo.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para ejecutar funciones de Lambda en contenedores, debe habilitar la versión 1 de cgroups, y habilitar y montar los cgroups de memoria y de dispositivos. Si no tiene previsto ejecutar funciones de Lambda en contenedores, puede omitir este paso.

   Para habilitar estas opciones de cgroups, arranque el dispositivo con los siguientes parámetros del kernel de Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obtener más información acerca de cómo ver y configurar los parámetros del kernel de su dispositivo, consulte la documentación del sistema operativo y del gestor de arranque. Siga las instrucciones para configurar permanentemente los parámetros del kernel.

5. Instale todas las demás dependencias necesarias en su dispositivo tal y como se indica en la lista de requisitos de Requisitos de los dispositivos.

Configuración de un dispositivo de Windows

> **Note**
>
> Esta característica está disponible para la versión 2.5.0 y versiones posteriores del componente núcleo de Greengrass.

Para configurar un dispositivo Windows para AWS IoT Greengrass V2

1. Instale el motor de ejecución de Java, que el software AWS IoT Greengrass principal necesita para ejecutarse. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.
2. Compruebe si Java está disponible en la variable del sistema PATH y agréguelo si no lo está. La LocalSystem cuenta ejecuta el software AWS IoT Greengrass principal, por lo que debe agregar Java a la variable de sistema PATH en lugar de a la variable de usuario PATH de su usuario. Haga lo siguiente:

   a. Pulse la tecla Windows para abrir el menú de inicio.

   b. Escriba **`environment variables`** para buscar las opciones del sistema en el menú de inicio.

c. En los resultados de la búsqueda del menú de inicio, elija Editar las variables de entorno del sistema para abrir la ventana de Propiedades del sistema.

d. Elija Variables de entorno... para abrir la ventana Variables de entorno.

e. En Variables del sistema, elija Ruta y, luego, Editar. En la ventana Editar variables de entorno, puede ver cada ruta en una línea independiente.

f. Compruebe si la ruta a la carpeta de la instalación de Java `bin` está presente. La ruta puede tener un aspecto similar al siguiente ejemplo.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Si la carpeta de la instalación de Java `bin` no aparece en Ruta, elija Nueva para agregarla y, a continuación, pulse Aceptar.

3. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.

4. Cree el usuario predeterminado en la LocalSystem cuenta del dispositivo Windows. *`password`*Sustitúyalo por una contraseña segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
6. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

   > **Note**
   >
   > En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información de usuario predeterminada en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Proporcione AWS las credenciales al dispositivo

Proporcione AWS las credenciales del dispositivo para que el instalador pueda aprovisionar los AWS recursos necesarios. Para obtener más información sobre los permisos necesarios, consulte Política de IAM mínima para que el instalador aprovisione recursos.

Para proporcionar AWS credenciales al dispositivo

- Proporcione sus AWS credenciales al dispositivo para que el instalador pueda aprovisionar los recursos de IAM AWS IoT y los de su dispositivo principal. Para aumentar la seguridad, le recomendamos que obtenga credenciales temporales para un rol de IAM que permita únicamente los permisos mínimos necesarios para el aprovisionamiento. Para obtener más información, consulte Política de IAM mínima para que el instalador aprovisione recursos.

> **Note**
>
> El instalador no guarda ni almacena sus credenciales.

En el dispositivo, realice una de las siguientes acciones para recuperar las credenciales y ponerlas a disposición del instalador del software AWS IoT Greengrass principal:

- (Recomendado) Utilice credenciales temporales de AWS IAM Identity Center

  a. Proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde IAM Identity Center. Para obtener más información, consulte la Actualización manual de credenciales en Obtener y actualizar las credenciales temporales en la Guía del usuario de IAM Identity Center.

  b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass principal.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- Use credenciales de seguridad temporales de un rol de (IAM):

  a. Proporcione el ID de clave de acceso, la clave de acceso secreta y el token de sesión desde un rol de IAM que asuma. Para obtener más información acerca de

cómo recuperar estas credenciales, consulte Solicitud de credenciales de seguridad temporales en la Guía del usuario de IAM.

b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciales a largo plazo de un usuario de IAM:

a. Proporcione el ID de clave de acceso y la clave de acceso secreta del usuario de IAM. Puede crear un usuario de IAM para el aprovisionamiento y luego eliminarlo. Para la política de IAM que debe proporcionarse al usuario, consulte Política de IAM mínima para que el instalador aprovisione recursos. Para obtener información acerca de cómo recuperar credenciales a largo plazo, consulte Administración de las claves de acceso de los usuarios de IAM en la Guía de usuario de IAM.

b. Ejecute los siguientes comandos para proporcionar las credenciales al software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

c. (Opcional) Si creó un usuario de IAM para aprovisionar su dispositivo de Greengrass, elimínelo.

d. (Opcional) Si utilizó el ID de clave de acceso y la clave de acceso secreta de un usuario de IAM existente, actualice las claves del usuario para que dejen de ser válidas. Para obtener más información, consulte Actualización de claves de acceso en la Guía de usuario de AWS Identity and Access Management .

Descargue el software AWS IoT Greengrass principal

Puede descargar la última versión del software AWS IoT Greengrass Core desde la siguiente ubicación:

- https://d2s8p88vqu9w66.cloudfront. net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Puede descargar una versión específica del software AWS IoT Greengrass Core desde la siguiente ubicación. *version*Sustitúyala por la versión que se va a descargar.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para descargar el software AWS IoT Greengrass principal

1. En su dispositivo principal, descargue el software AWS IoT Greengrass Core en un archivo denominado`greengrass-nucleus-latest.zip`.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Al descargar este software, acepta el acuerdo de licencia del software de Greengrass Core.

2. (Opcional) Verificación de la firma del software del núcleo de Greengrass

> **Note**
>
> Esta característica está disponible en la versión 2.9.5 y versiones posteriores del núcleo de Greengrass.

a. Use el siguiente comando para verificar la firma del artefacto del núcleo de Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

El nombre del archivo puede tener un aspecto diferente según la versión de JDK que instale. Reemplace *jdk17.0.6_10* por la versión de JDK que instaló.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. La invocación `jarsigner` produce un resultado que indica los resultados de la verificación.

i. Si el archivo zip del núcleo de Greengrass está firmado, el resultado contiene la siguiente declaración:

```
jar verified.
```

ii. Si el archivo zip del núcleo de Greengrass no está firmado, el resultado contiene la siguiente declaración:

```
jar is unsigned.
```

c. Si ha proporcionado la opción `-certs` Jarsigner junto con las opciones `-verify` y `-verbose`, el resultado también incluye información detallada del certificado de firmante.

3. Descomprime el software AWS IoT Greengrass Core en una carpeta de tu dispositivo. *GreengrassInstaller*Sustitúyalo por la carpeta que desee usar.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
```

```
rm greengrass-nucleus-latest.zip
```

4. (Opcional) Ejecute el siguiente comando para ver la versión del software AWS IoT Greengrass principal.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Si instala una versión del núcleo de Greengrass anterior a la v2.4.0, no elimine esta carpeta después de instalar el software Core. AWS IoT Greengrass El software AWS IoT Greengrass Core utiliza los archivos de esta carpeta para ejecutarse.
> Si descargó la última versión del software, instale la versión 2.4.0 o posterior y podrá eliminar esta carpeta después de instalar el software AWS IoT Greengrass principal.

Instale el software principal AWS IoT Greengrass

Ejecute el instalador con argumentos que especifiquen lo siguiente:

- Cree los AWS recursos que el dispositivo principal necesita para funcionar.
- Especifique si desea usar el usuario del sistema `ggc_user` para ejecutar los componentes de software en el dispositivo principal. En los dispositivos Linux, este comando también especifica el uso del grupo del sistema `ggc_group` y el instalador crea el usuario y el grupo del sistema por usted.
- Configure el software AWS IoT Greengrass Core como un servicio del sistema que se ejecute durante el arranque. En los dispositivos Linux, esto requiere el sistema de inicio Systemd.

> **Important**
>
> En los dispositivos principales de Windows, debe configurar el software AWS IoT Greengrass principal como un servicio del sistema.

Para configurar un dispositivo de desarrollo con herramientas de desarrollo local, especifique el argumento `--deploy-dev-tools true`. Una vez finalizada la instalación, la implementación de las herramientas de desarrollo local puede tardar hasta un minuto.

Para obtener más información acerca de los argumentos que puede especificar, consulte Argumentos del instalador.

> **Note**
>
> Si utilizas un AWS IoT Greengrass dispositivo con memoria limitada, puedes controlar la cantidad de memoria que utiliza el software AWS IoT Greengrass Core. Para controlar la asignación de memoria, puede configurar las opciones de tamaño de montón de la JVM en el parámetro de configuración `jvmOptions` del componente núcleo. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

Para instalar el software AWS IoT Greengrass Core

1. Use un editor de texto para crear un archivo de configuración llamado `config.yaml` para proporcionárselo al instalador.

   Por ejemplo, en un sistema basado en Linux, puede ejecutar el siguiente comando para usar GNU nano a fin de crear el archivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie el siguiente contenido YAML en el archivo. Este archivo de configuración parcial especifica los parámetros del sistema y los parámetros del núcleo de Greengrass.

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       configuration:
         fipsMode: "true"
         iotDataEndpoint: "data.iot-fips.us-west-2.amazonaws.com"
         iotCredEndpoint: "data.credentials.iot-fips.us-west-2.amazonaws.com"
         greengrassDataPlaneEndpoint: "iotData"
   ```

   - *us-west-2*Sustitúyalo por el Región de AWS lugar donde creó los recursos.
   - Reemplace el *iotDataEndpoint* por su punto de conexión de datos de AWS IoT .
   - Reemplace el *iotCredEndpoint* por su punto de conexión de credenciales de AWS IoT .

2. Ejecute el instalador AWS IoT Greengrass principal. Reemplace los valores de los argumentos en su comando de la siguiente manera.

> **Note**
>
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si usa Windows, use una carpeta raíz como `C:\greengrass\v2` o `D:\greengrass\v2` para mantener las rutas de los componentes de Greengrass por debajo del límite de 260 caracteres.

a. *`/greengrass/v2`*o bien*`C:\greengrass\v2`*: la ruta a la carpeta raíz que se utilizará para instalar el software AWS IoT Greengrass Core.

b. *`GreengrassInstaller`*. La ruta a la carpeta en la que desempaquetó el instalador del software AWS IoT Greengrass Core.

c. *`region`*. El Región de AWS lugar en el que encontrar o crear recursos.

d. *`MyGreengrassCore`*. El nombre del AWS IoT dispositivo principal de Greengrass. Si el objeto no existe, el instalador la crea. El instalador descarga los certificados para autenticarse como tal. AWS IoT Para obtener más información, consulte Autenticación y autorización de dispositivos para AWS IoT Greengrass.

> **Note**
>
> El nombre del objeto no puede contener dos puntos (`:`).

e. *`MyGreengrassCoreGroup`*. El nombre del grupo de AWS IoT cosas de su dispositivo principal de Greengrass. Si el grupo de objetos no existe, el instalador lo crea y le agrega un objeto. Si el grupo de objetos existe y tiene una implementación activa, el dispositivo principal descarga y ejecuta el software que especifique la implementación.

> **Note**
>
> El nombre del grupo de objetos no puede contener dos puntos (`:`).

f. *`GreengrassV2IoTThingPolicy`*. El nombre de la AWS IoT política que permite a los dispositivos principales de Greengrass comunicarse con AWS IoT y. AWS IoT Greengrass Si la AWS IoT política no existe, el instalador crea una AWS IoT política permisiva con este nombre. Puede restringir los permisos de esta política según su caso de uso. Para

obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

g. *GreengrassV2TokenExchangeRole*. El nombre de la función de IAM que permite al dispositivo principal de Greengrass obtener AWS credenciales temporales. Si el rol no existe, el instalador lo crea y asocia una política denominada *GreengrassV2TokenExchangeRole*`Access`. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

h. *GreengrassCoreTokenExchangeRoleAlias*. El alias de la función de IAM que permite al dispositivo principal de Greengrass obtener credenciales temporales más adelante. Si el alias del rol no existe, el instalador lo crea y lo dirige al rol de IAM que especifique. Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --init-config ./GreengrassInstaller/config.yaml \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
```

```
  --provision true ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true
```

> **Important**
>
> En los dispositivos principales de Windows, debe especificar si `--setup-system-service true` desea configurar el software AWS IoT Greengrass Core como un servicio del sistema.

El instalador imprime los siguientes mensajes si la operación es exitosa:

- Si especifica `--provision`, el instalador imprime `Successfully configured Nucleus with provisioned resource details` si configuró los recursos correctamente.
- Si especifica `--deploy-dev-tools`, el instalador imprime `Configured Nucleus to deploy aws.greengrass.Cli component` si creó la implementación correctamente.
- Si especifica `--setup-system-service true`, el instalador imprime `Successfully set up Nucleus as a system service` si configuró y ejecutó el software como un servicio.
- Si no especifica `--setup-system-service true`, el instalador imprime `Launched Nucleus successfully` si se ejecutó correctamente y ejecutó el software.

3. Omita este paso si instaló la versión 2.0.4 o una versión posterior de Núcleo de Greengrass. Si descargó la versión más reciente del software, instaló la versión 2.0.4 o una versión posterior.

Ejecute el siguiente comando para establecer los permisos de archivo necesarios para la carpeta raíz del software AWS IoT Greengrass Core. */greengrass/v2*Sustitúyala por la carpeta raíz que especificó en el comando de instalación y */greengrass* sustitúyala por la carpeta principal de la carpeta raíz.

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

Si instaló el software AWS IoT Greengrass principal como un servicio del sistema, el instalador ejecutará el software automáticamente. De no ser así, debe ejecutar el software manualmente. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.

> **Note**
>
> De forma predeterminada, el rol de IAM que crea el instalador no permite el acceso a los artefactos de componentes de los buckets de S3. Para implementar componentes personalizados que definan artefactos en Amazon S3, debe agregar permisos al rol para permitir que su dispositivo principal recupere artefactos de componentes. Para obtener más información, consulte Cómo permitir el acceso a los buckets de S3 para los artefactos del componente.
>
> Si aún no tiene un bucket de S3 para los artefactos de los componentes, puede agregar estos permisos más adelante, después de crear un bucket.

Para obtener más información sobre cómo configurar y utilizar el software AWS IoT Greengrass, consulte lo siguiente:

- Configurar el software AWS IoT Greengrass principal
- Desarrollo de componentes de AWS IoT Greengrass
- Implemente AWS IoT Greengrass componentes en los dispositivos
- Interfaz de la línea de comandos de Greengrass

Componentes propios que cumplen con las normas de FIPS

aws.greengrass.Nucleus	data.iot-fips. *us-east-1* .amazonaws.com
	greengrass-fips. *us-east-1* .amazonaws.com
	data.credentials.iot-fips. *us-east-1* .amazonaws.com
aws.greengrass.TokenExchangeService	data.credentials.iot-fips. *us-east-1* .amazonaws.com
aws.greengrass.Cli	
aws.greengrass.StreamManager	• kinesis-fips. *us-east-1* .amazonaws.com • data.iotsitewise-fips. *us-east-1* .amazonaws.com • s3-fips.*us-east-1* .amazonaws.com **Note** El administrador de flujos no es compatible con el punto AWS IoT Analytics final FIPS
aws.greengrass.LogManager	logs-fips. *us-east-1* .amazonaws.com
aws.greengrass.crypto.Pkcs11Provider	
aws.greengrass.ShadowManager	

`aws.greengrass.DockerApplicationManager`	consejos electrónicos. *us-east-1* .amazonaws.com
`aws.greengrass.SecretManager`	administrador de secretos - fips. *us-east-1* .amazonaws.com
`aws.greengrass.telemetry.NucleusEmitter`	
`aws.greengrass.clientdevices.IPDetector`	
`aws.greengrass.DiskSpooler`	

Resiliencia en AWS IoT Greengrass

La infraestructura global de AWS se basa en regiones y zonas de disponibilidad de Amazon Web Services. Cada región de Región de AWS proporciona varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos.

Para obtener más información, consulte Infraestructura global de AWS.

Además de la infraestructura global de AWS, AWS IoT Greengrass ofrece varias características que le ayudan con sus necesidades de resiliencia y copia de seguridad de los datos.

- Puede configurar un dispositivo principal de Greengrass para escribir registros en el sistema de archivos local y en los Registros de CloudWatch. Si el dispositivo principal pierde la conectividad, puede seguir registrando mensajes en el sistema de archivos. Cuando se vuelve a conectar, escribe los mensajes de registro en los Registros de CloudWatch. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.
- Si un dispositivo principal se queda sin alimentación durante una implementación, se reanudará la implementación cuando se reinicie el software AWS IoT Greengrass Core.

- Si el dispositivo principal pierde conectividad a Internet, los dispositivos de cliente de Greengrass pueden seguir comunicándose a través de la red local.
- Puede crear componentes de Greengrass que lean flujos del administrador de flujos y envíen los datos a destinos de almacenamiento locales.

Seguridad de la infraestructura en AWS IoT Greengrass

Como servicio gestionado, AWS IoT Greengrass está protegido por los procedimientos de seguridad de red AWS global que se describen en el documento técnico Amazon Web Services: Overview of Security Processes.

Utiliza las llamadas a la API AWS publicadas para acceder a AWS IoT Greengrass través de la red. Los clientes deben ser compatibles con Transport Layer Security (TLS) 1.2 o con una versión posterior. Recomendamos TLS 1.3 o una versión posterior. Los clientes también deben ser compatibles con conjuntos de cifrado con confidencialidad directa total (PFS), como Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

En un AWS IoT Greengrass entorno, los dispositivos utilizan certificados X.509 y claves criptográficas para conectarse y autenticarse en él. Nube de AWS Para obtener más información, consulte the section called “Autenticación y autorización de dispositivos”.

Análisis de configuración y vulnerabilidad en AWS IoT Greengrass

Los entornos de IoT pueden constar de un gran número de dispositivos que tienen diversas capacidades, son de larga duración y están distribuidos geográficamente. Estas características hacen que la configuración del dispositivo sea compleja y propensa a errores. Y dado que los dispositivos a menudo están limitados en potencia informática, memoria y capacidades de almacenamiento, esto limita el uso del cifrado y otras formas de seguridad en los propios dispositivos. Además, los dispositivos a menudo usan software con vulnerabilidades conocidas. Estos factores hacen que los dispositivos de IoT sean un objetivo atractivo para los piratas informáticos y dificultan la protección continuada de los mismos.

AWS IoT Device Defender aborda estos desafíos proporcionando herramientas para identificar los problemas de seguridad y las desviaciones de las mejores prácticas. Puede utilizarlos AWS IoT Device Defender para analizar, auditar y supervisar los dispositivos conectados a fin de detectar comportamientos anormales y mitigar los riesgos de seguridad. AWS IoT Device Defender puede auditar los dispositivos para asegurarse de que cumplen con las mejores prácticas de seguridad y detectar un comportamiento anormal en los dispositivos. Esto permite aplicar políticas de seguridad coherentes en todos sus dispositivos y responder rápidamente cuando los dispositivos se ven comprometidos. IForpara obtener más información, consulte los siguientes temas:

- El componente Device Defender
- AWS IoT Device Defender en la Guía para desarrolladores de AWS IoT Core .

En AWS IoT Greengrass los entornos, debe tener en cuenta las siguientes consideraciones:

- Es su responsabilidad proteger los dispositivos físicos, el sistema de archivos en sus dispositivos y la red local.
- AWS IoT Greengrass no impone el aislamiento de red para los componentes de Greengrass definidos por el usuario, se ejecuten o no en un contenedor de Greengrass. Por lo tanto, es posible que los componentes de Greengrass se comuniquen con cualquier otro proceso que se ejecute en el sistema o fuera a través de la red.

Integridad del código en AWS IoT Greengrass V2

AWS IoT Greengrass implementa componentes de software desde la Nube de AWS a dispositivos que ejecutan el software AWS IoT Greengrass Core. Estos componentes de software incluyen los componentes proporcionados por AWS y los componentes personalizados que puede cargar en su Cuenta de AWS. Cada componente incluye una receta. La receta define los metadatos del componente y cualquier cantidad de artefactos, que son componentes binarios, como el código compilado y los recursos estáticos. Los artefactos de los componentes se almacenan en Amazon S3.

A medida que desarrolle e implemente los componentes de Greengrass, siga estos pasos básicos que funcionan con los artefactos de los componentes en su Cuenta de AWS y en sus dispositivos:

1. Cree y cargue artefactos en buckets de S3.
2. Cree un componente a partir de una receta y artefactos en el servicio de AWS IoT Greengrass, que calcula un hash criptográfico de cada artefacto.

3. Implemente un componente en los dispositivos principales de Greengrass, que descargan y verifican la integridad de cada artefacto.

AWS es responsable de mantener la integridad de los artefactos después de cargarlos en buckets de S3, incluso cuando implementa componentes en los dispositivos principales de Greengrass. Usted es responsable de proteger los artefactos de software antes de subirlos a los buckets de S3. También es responsable de proteger el acceso a los recursos en su Cuenta de AWS, incluidos los buckets de S3 en los que carga los artefactos del componente.

> **Note**
>
> Amazon S3 ofrece una característica llamada bloqueo de objetos de S3 que puede usar para protegerse contra los cambios en los artefactos del componente en los buckets de S3 de su Cuenta de AWS. Puede usar el bloqueo de objetos de S3 para evitar que se eliminen o se sobrescriban los artefactos del componente. Para obtener más información, consulte el uso del bloqueo de objetos de S3 en la Guía del usuario de Amazon Simple Storage Service.

Cuando AWS publica un componente público y cuando carga un componente personalizado, AWS IoT Greengrass calcula un resumen criptográfico para cada artefacto del componente. AWS IoT Greengrass actualiza la receta del componente para incluir el resumen de cada artefacto y el algoritmo hash usado para calcular ese resumen. Este resumen garantiza la integridad del artefacto, ya que si el artefacto cambia durante en la Nube de AWS o durante la descarga, el resumen del archivo no coincidirá con el resumen que AWS IoT Greengrass almacena en la receta del componente. Para obtener más información, consulte los artefactos en la referencia de la receta del componente.

Al implementar un componente en un dispositivo principal, el software principal AWS IoT Greengrass descarga la receta del componente y cada artefacto del componente que define la receta. El software principal de AWS IoT Greengrass calcula el resumen de cada archivo de artefacto descargado y lo compara con el resumen de ese artefacto de la receta. Si los resúmenes no coinciden, se produce un error en la implementación y el software AWS IoT Greengrass Core elimina los artefactos descargados del sistema de archivos del dispositivo. Para obtener más información sobre cómo se protegen las conexiones entre los dispositivos principales y AWS IoT Greengrass, consulte Cifrado en tránsito.

Usted es responsable de proteger los archivos del artefacto de componente en los sistemas de archivos del dispositivo principal. El software principal de AWS IoT Greengrass guarda los artefactos

en la carpeta `packages` en la carpeta raíz de Greengrass. Puede usar AWS IoT Device Defender para analizar, auditar y monitorear los dispositivos principales. Para obtener más información, consulte Análisis de configuración y vulnerabilidad en AWS IoT Greengrass.

AWS IoT Greengrass y puntos finales de VPC de interfaz ()AWS PrivateLink

Puede establecer una conexión privada entre la VPC y el plano de AWS IoT Greengrass control mediante la creación de un punto final de la VPC de interfaz. Puede usar este punto final para administrar los componentes, las implementaciones y los dispositivos principales del servicio. AWS IoT Greengrass Los puntos finales de la interfaz funcionan con una tecnología que le permite acceder de AWS IoT Greengrass APIs forma privada sin una puerta de enlace a Internet, un dispositivo NAT, una conexión VPN o una conexión AWS Direct Connect. AWS PrivateLink Las instancias de su VPC no necesitan direcciones IP públicas para comunicarse con ellas. AWS IoT Greengrass APIs El tráfico entre tu VPC y AWS IoT Greengrass no sale de la red de Amazon.

Cada punto de conexión de la interfaz está representado por una o más interfaces de red elásticas en las subredes.

Para obtener más información, consulte Puntos de conexión de VPC de interfaz (AWS PrivateLink) en la Guía del usuario de Amazon VPC.

Temas

- Consideraciones sobre los puntos AWS IoT Greengrass finales de VPC
- Cree un punto final de VPC de interfaz para las operaciones del plano AWS IoT Greengrass de control
- Crear una política de puntos de conexión de VPC para AWS IoT Greengrass
- Opere un dispositivo AWS IoT Greengrass central en VPC

Consideraciones sobre los puntos AWS IoT Greengrass finales de VPC

Antes de configurar un punto de enlace de VPC de interfaz AWS IoT Greengrass, consulte las propiedades y limitaciones del punto de enlace de interfaz en la Guía del usuario de Amazon VPC. Además, tenga en cuenta las siguientes consideraciones:

- AWS IoT Greengrass admite realizar llamadas a todas las acciones de la API de su plano de control desde su VPC. El plano de control incluye operaciones como

CreateDeployment y ListEffectiveDeployments. El plano de control no incluye operaciones como ResolveComponentCandidatesDiscover, que son operaciones del plano de datos.

- Los puntos de enlace de VPC para actualmente no AWS IoT Greengrass se admiten en las regiones de China AWS .

Cree un punto final de VPC de interfaz para las operaciones del plano AWS IoT Greengrass de control

Puede crear un punto final de VPC para el plano de AWS IoT Greengrass control mediante la consola Amazon VPC o el (). AWS Command Line Interface AWS CLI Para obtener más información, consulte Creación de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Cree un punto final de VPC para AWS IoT Greengrass usar el siguiente nombre de servicio:

- com.amazonaws. *region*.greengrass

Si habilita el DNS privado para el punto final, puede realizar solicitudes a la API para AWS IoT Greengrass utilizar su nombre de DNS predeterminado para la región, por ejemplo. `greengrass.us-east-1.amazonaws.com` El DNS privado está habilitado de forma predeterminada.

Para más información, consulte Acceso a un servicio a través de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Crear una política de puntos de conexión de VPC para AWS IoT Greengrass

Puede adjuntar una política de punto de conexión a su punto de conexión de VPC que controle el acceso a las operaciones del plano de control de AWS IoT Greengrass . La política especifica la siguiente información:

- La entidad principal que puede realizar acciones.
- Acciones que la entidad principal puede realizar.
- Los recursos sobre los que la entidad principal puede realizar acciones.

Para más información, consulte Control del acceso a los servicios con puntos de conexión de VPC en la Guía del usuario de Amazon VPC.

Example Ejemplo: política de puntos finales de VPC para acciones AWS IoT Greengrass

El siguiente es un ejemplo de una política de puntos finales para AWS IoT Greengrass. Cuando se adjunta a un punto final, esta política otorga acceso a las AWS IoT Greengrass acciones enumeradas a todos los principales de todos los recursos.

```
{
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:ListEffectiveDeployments"
            ],
            "Resource": "*"
        }
    ]
}
```

Opere un dispositivo AWS IoT Greengrass central en VPC

Puede operar un dispositivo principal de Greengrass y realizar implementaciones en VPC sin acceso público a Internet. Como mínimo, debe configurar los siguientes puntos de conexión de VPC con los alias de DNS correspondientes. Para obtener más información sobre cómo crear y utilizar puntos de conexión de VPC, consulte Crear un punto de conexión de VPC en la Guía del usuario de Amazon VPC.

> **Note**
>
> La función de VPC para crear automáticamente un registro DNS está deshabilitada para las credenciales AWS IoT data y AWS IoT . Para conectarse a estos puntos de conexión, debe crear manualmente un registro DNS privado. Para obtener más información, consulte DNS privado para puntos de conexión de interfaz. Para obtener más información sobre las limitaciones de la AWS IoT Core VPC, consulte Limitaciones de los puntos finales de la VPC.

Requisitos previos

- Debe instalar el software AWS IoT Greengrass principal siguiendo los pasos de aprovisionamiento manual. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos.

Limitaciones

- El funcionamiento de un dispositivo principal de Greengrass en VPC no es compatible en las regiones de China y AWS GovCloud (US) Regions.
- Para obtener más información sobre las limitaciones de los puntos de AWS IoT data enlace de VPC del proveedor de AWS IoT credenciales, consulte Limitaciones.

Configuración de su dispositivo principal de Greengrass para que funcione en VPC

1. Obtenga los AWS IoT puntos de conexión que desee y Cuenta de AWS guárdelos para usarlos más adelante. El dispositivo usa estos puntos de conexión para conectarse a AWS IoT. Haga lo siguiente:

   a. Obtenga el punto final AWS IoT de datos para su. Cuenta de AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   b. Obtenga el punto final de AWS IoT credenciales para su Cuenta de AWS.

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   Si la solicitud se realiza exitosamente, la respuesta se parece al siguiente ejemplo.

   ```
   {
   ```

```
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

2. Cree una interfaz de Amazon VPC para los puntos de enlace AWS IoT data y AWS IoT las credenciales:

   a. Vaya a la consola de puntos de conexión de VPC, en Nube virtual privada en el menú de la izquierda, elija Puntos de enlace y después Crear punto de conexión.

   b. En la página Crear punto de conexión, especifique la siguiente información.

   - Elija Servicio de AWS en Categoría de servicio.
   - En Nombre del servicio, busque introduciendo la palabra clave `iot`. En la lista de servicios de `iot` que se muestra, elija el punto de conexión.

     Si crea un punto de enlace de VPC para el plano de AWS IoT Core datos, elija el punto de enlace de la API del plano de AWS IoT Core datos para su región. El punto de conexión tendrá el formato `com.amazonaws.`*`region`*`.iot.data`.

     Si crea un punto de enlace de VPC para el proveedor de AWS IoT Core credenciales, elija el punto de enlace del proveedor de AWS IoT Core credenciales para su región. El punto de conexión tendrá el formato `com.amazonaws.`*`region`*`.iot.credentials`.

     > **Note**
     >
     > El nombre del servicio para el plano de AWS IoT Core datos en la región de China tendrá este formato`cn.com.amazonaws.`*`region`*`.iot.data`. La región de China no admite la creación de puntos finales de VPC para el proveedor de AWS IoT Core credenciales.

   - Para la VPC y las subredes, elija la VPC en la que desee crear el punto final y las zonas de disponibilidad (AZs) en las que desee crear la red del punto final.
   - En Habilitar nombre de DNS, asegúrese de que Habilitar para este punto de conexión no está seleccionado. Ni el plano AWS IoT Core de datos ni el proveedor de AWS IoT Core credenciales admiten todavía nombres DNS privados.
   - En Grupo de seguridad, elija los grupos de seguridad que deban asociarse a las interfaces de red de punto de conexión.

- Puede agregar o eliminar etiquetas si lo desea. Las etiquetas son pares de nombre-valor que se utilizan para asociar al punto de conexión.

c. Para crear su punto de conexión de VPC, elija Crear punto de conexión.

3. Después de crear el AWS PrivateLink punto final, en la pestaña Detalles del dispositivo, verá una lista de nombres de DNS. Puede utilizar uno de estos nombres DNS que ha creado en esta sección para configurar su zona alojada privada.
4. Cree un punto de conexión de Amazon S3. Para obtener más información consulte Crear un punto de conexión de VPC de Amazon S3.
5. Si utiliza los componentes de Greengrass proporcionados por AWS, es posible que se necesiten configuraciones y puntos de conexión adicionales. Para ver los requisitos de los puntos de conexión, seleccione el componente de la lista de componentes proporcionados por AWS y consulte la sección de requisitos. Por ejemplo, los requisitos del componente del administrador de registros indican que este componente debe poder realizar las solicitudes salientes al punto de conexión `logs.region.amazonaws.com`.

   Si utiliza su propio componente, es posible que deba revisar las dependencias y realizar pruebas adicionales para determinar si se requieren puntos de conexión adicionales.
6. En la configuración del núcleo de Greengrass, `greengrassDataPlaneEndpoint` debe estar configurado en **`iotdata`**. Para obtener más información, consulte la Configuración del núcleo de Greengrass.
7. Si se encuentra en la región `us-east-1`, defina el parámetro de configuración `s3EndpointType` en **`REGIONAL`** en la configuración del núcleo de Greengrass. Esta característica está disponible para las versiones 2.11.3 y posteriores del núcleo de Greengrass.

Example Ejemplo: configuración de componentes

```
{
"aws.greengrass.Nucleus": {
   "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
     }
   }
```

```
}
```

En la siguiente tabla, se proporciona información sobre los alias de DNS privados personalizados correspondientes.

Servicio	Nombre del servicio de punto de conexión de VPC	Tipo de punto de conexión de VPC	Alias de DNS privado personalizado	Notas
AWS IoT data	`com.amazonaws.`*`region`*`.iot.d`	Interfaz	*`prefix`*`-ats.iot.`*`region`*`.a` `s.com`	El registro DNS privado debe coincidir con el AWS IoT data punto final de su cuenta:`aws iot describe-endpoint --endpoint-type iot:Data-ATS .`

AWS IoT Credenciales	`com.amazonaws.`*`region`*`.iot.c`
	`entials`
		Interfaz	*`prefix`*`.c`
			`als.iot.`
			`s.com`
				El registro DNS privado

Servicio	Nombre del servicio de punto de conexión de VPC	Tipo de punto de conexión de VPC	Alias de DNS privado personalizado	Notas
				debe coincidir con el punto final de AWS IoT las credenciales de su cuenta:`aws iot describe-endpoint --endpoint-type iot:CredentialProvider .`
Amazon S3	`com.amazonaws.` *`region`*`.s3`	Interfaz		El registro DNS se crea automáticamente.

Mejores prácticas de seguridad para AWS IoT Greengrass

Este tema contiene las mejores prácticas de seguridad para AWS IoT Greengrass.

Conceda los mínimos permisos posibles

Siga el principio de privilegio mínimo para los componentes ejecutándolos como usuarios sin privilegios. Los componentes no deben ejecutarse como raíz a menos que sea absolutamente necesario.

Utilice el conjunto mínimo de permisos en roles de IAM. Limite el uso del comodín * para las propiedades `Action` y `Resource` en las políticas de IAM. En su lugar, declare un conjunto finito de acciones y recursos cuando sea posible. Para obtener más información acerca de los privilegios mínimos y otras prácticas recomendadas de política, consulte the section called “Prácticas recomendadas sobre las políticas”.

La mejor práctica de privilegios mínimos también se aplica a AWS IoT las políticas que asocie a su núcleo de Greengrass.

No codifique las credenciales en los componentes de Greengrass

No codifique las credenciales en los componentes de Greengrass definidos por el usuario. Para proteger mejor sus credenciales:

- Para interactuar con AWS los servicios, defina los permisos para acciones y recursos específicos en la función de servicio principal de dispositivos de Greengrass.
- Utilice el componente de administrador de secretos para almacenar sus credenciales. O bien, si la función usa el AWS SDK, use las credenciales de la cadena de proveedores de credenciales predeterminada.

No registre información confidencial

Debe evitar el registro de credenciales y otra información de identificación personal (PII). Le recomendamos que implemente las siguientes medidas de seguridad, aunque el acceso a los registros locales en un dispositivo principal requiera privilegios de root y el acceso a los CloudWatch registros requiera permisos de IAM.

- No utilice información confidencial en las rutas de temas de MQTT.
- No utilice información confidencial en nombres, tipos y atributos de dispositivo (objeto) en el registro de AWS IoT Core .
- No registre información confidencial en los componentes de Greengrass o funciones de lambda definidas por el usuario.

- No utilice información confidencial en los nombres ni en los IDs recursos de Greengrass:
  - Dispositivos principales
  - Componentes
  - Implementaciones
  - Loggers

Mantenga sincronizado el reloj del dispositivo

Es importante que la hora del dispositivo sea precisa. Los certificados X.509 tienen una fecha y una hora de caducidad. El reloj del dispositivo se utiliza para comprobar que un certificado de servidor sigue siendo válido. Los relojes de dispositivos pueden variar con el tiempo o las baterías pueden descargarse.

Para obtener más información, consulte las prácticas recomendadas Mantener sincronizado el reloj del dispositivo en la Guía del desarrollador de AWS IoT Core .

Recomendaciones de conjunto de cifrado

Greengrass selecciona de forma predeterminada los conjuntos de cifrado TLS más recientes disponibles en el dispositivo. Considere deshabilitar el uso de conjuntos de cifrado antiguos en el dispositivo. Por ejemplo, los conjuntos de cifrado CBC.

Para obtener más información, consulte la configuración de cifrado de Java.

Véase también

- Prácticas recomendadas de seguridad para AWS IoT Core en la Guía del desarrollador de AWS IoT
- Diez reglas de oro de seguridad para las soluciones de IoT industrial en el Internet de las cosas en el blog AWS oficial

Uso de AWS IoT Device Tester para la versión 2 de AWS IoT Greengrass

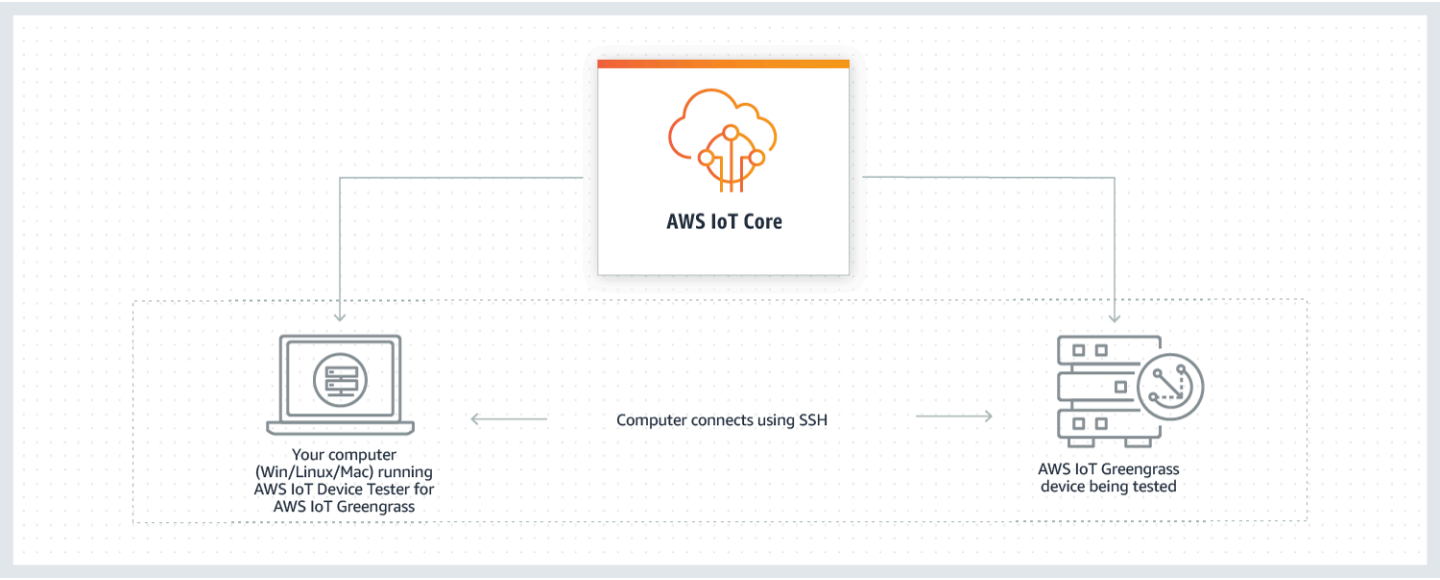
AWS IoT Device Tester (IDT) es un marco de pruebas descargable que le permite validar dispositivos de IoT. Puede usar IDT de AWS IoT Greengrass para ejecutar el conjunto de calificaciones de AWS IoT Greengrass y crear y ejecutar conjuntos de pruebas personalizados para sus dispositivos.

IDT para AWS IoT Greengrass se ejecuta en su equipo host (Windows, MacOS o Linux) conectado al dispositivo que se tiene que probar. Ejecuta pruebas y agrega resultados. También proporciona una interfaz de línea de comandos para administrar el proceso de pruebas.

Guía de cualificación de AWS IoT Greengrass

Utilice AWS IoT Device Tester para la versión 2 de AWS IoT Greengrass para comprobar que el software AWS IoT Greengrass Core se ejecuta en su hardware y puede comunicarse con la Nube de AWS. También realiza pruebas integrales con AWS IoT Core. Por ejemplo, verifica que su dispositivo pueda implementar componentes y actualizarlos.

Si desea añadir su hardware al catálogo de dispositivos AWS Partner, ejecute el conjunto de cualificación AWS IoT Greengrass para generar informes de pruebas que puede enviar a AWS IoT. Para obtener más información, consulte el Programa de Calificación de Dispositivos de AWS.



IDT para la versión 2 de AWS IoT Greengrass organiza pruebas mediante los conceptos de conjunto de pruebas y grupos de pruebas.

- Un conjunto de pruebas es el conjunto de grupos de pruebas que se utiliza para verificar que un dispositivo funciona con versiones particulares de AWS IoT Greengrass.
- Un grupo de pruebas es el conjunto de pruebas individuales relacionadas con una característica concreta, como implementaciones de componentes.

Para obtener más información, consulte Utilice IDT para ejecutar el conjunto de AWS IoT Greengrass cualificaciones.

Compatibilidad con los conjuntos de prueba

A partir de la versión 4.0.1 de IDT, IDT para AWS IoT Greengrass V2 combina una configuración y un formato de resultados estandarizados con un entorno de conjuntos de pruebas que le permite desarrollar conjuntos de pruebas personalizados para sus dispositivos y su software. Puede agregar pruebas personalizadas para su propia validación interna o proporcionárselas a sus clientes para la verificación de los dispositivos.

La forma en que un escritor de pruebas configura un conjunto de pruebas personalizado determina las configuraciones de configuración necesarias para ejecutar conjuntos de pruebas personalizados. Para obtener más información, consulte Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas.

Versiones compatibles de AWS IoT Device Tester para AWS IoT Greengrass V2

En este tema se enumeran las versiones compatibles de IDT para V2. AWS IoT Greengrass
Como práctica recomendada, le recomendamos que utilice la última versión de IDT para AWS IoT Greengrass V2 que sea compatible con la versión de destino de la AWS IoT Greengrass V2. Las nuevas versiones de AWS IoT Greengrass pueden requerir la descarga de una nueva versión de IDT para AWS IoT Greengrass V2. Cuando inicie una prueba, recibirá una notificación si IDT para AWS IoT Greengrass V2 no es compatible con la versión AWS IoT Greengrass que está utilizando.

Al descargar el software, acepta el Acuerdo de licencia de AWS IoT Device Tester.

> **Note**
>
> IDT no admite la ejecución por parte de varios usuarios desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.

Última versión de IDT para V2 AWS IoT Greengrass

Puede usar esta versión de IDT para AWS IoT Greengrass V2 con la AWS IoT Greengrass versión que se indica aquí.

IDT v4.9.4 para AWS IoT Greengrass

Versiones compatibles: AWS IoT Greengrass

- Versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 del núcleo de Greengrass

Descargas de software de IDT:

- IDT v4.9.4 con el conjunto GGV2 de pruebas Q_2.5.4 para Linux
- IDT v4.9.4 con el conjunto GGV2 de pruebas Q_2.5.4 para macOS
- IDT v4.9.4 con el conjunto de pruebas Q_2.5.4 para Windows GGV2

Notas de la versión:

- Permite la validación y calificación de dispositivos que ejecutan las versiones de software AWS IoT Greengrass Core 2.12.0, 2.11.0, 2.10.0 y 2.9.5.
- Elimina los grupos de pruebas del administrador de flujos y machine learning.

Notas adicionales:

- Si su dispositivo usa un HSM y usted usa un núcleo 2.10.x, migre a la versión 2.11.0 o posterior del núcleo de Greengrass.

Versión del conjunto de pruebas:

`GGV2Q_2.5.4`

- Publicado el 03 de mayo de 2024

Versiones anteriores de IDT para AWS IoT Greengrass

También se admiten las siguientes versiones anteriores de IDT para AWS IoT Greengrass V2.

IDT v4.9.3 para AWS IoT Greengrass

Versiones compatibles: AWS IoT Greengrass

- Versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 del núcleo de Greengrass

Descargas de software de IDT:

- IDT v4.9.3 con el conjunto GGV2 de pruebas Q_2.5.3 para Linux
- IDT v4.9.3 con el conjunto GGV2 de pruebas Q_2.5.3 para macOS
- IDT v4.9.3 con el conjunto de pruebas Q_2.5.3 para Windows GGV2

Notas de la versión:

- Corrige un problema en las pruebas de componentes al probar un dispositivo Linux desde un host Windows o viceversa.
- Elimina el caso de prueba `localcomponent` del grupo de pruebas `component`. Este caso de pruebas ya no es obligatorio para la calificación.

Notas adicionales:

- Si su dispositivo usa un HSM y usted usa un núcleo 2.10.x, migre a la versión 2.11.0 o posterior del núcleo de Greengrass.

Versión del conjunto de pruebas:

`GGV2Q_2.5.3`

- Publicado el 05 de abril de 2024

Versiones no AWS IoT Device Tester compatibles AWS IoT Greengrass de para V2

En este tema se enumeran las versiones no compatibles de IDT para la versión 2. AWS IoT Greengrass Las versiones que no son compatibles no reciben actualizaciones ni correcciones de errores. Para obtener más información, consulte the section called “Política de soporte de AWS IoT Device Tester para AWS IoT Greengrass”.

IDT v4.9.2 para AWS IoT Greengrass

Notas de la versión:

- Soluciona un problema por el que el conjunto de pruebas de Lambda fallaba debido a la obsolescencia de Java 8.

Versión del conjunto de pruebas:

`GGV2Q_2.5.2`

- Publicado el 18 de marzo de 2024

IDT v4.9.1 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan las versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 del software AWS IoT Greengrass Core.
- Correcciones de errores menores.

Versión del conjunto de pruebas:

`GGV2Q_2.5.1`

- Publicado el 05 de octubre de 2023

IDT v4.7.0 para AWS IoT Greengrass

Versiones compatibles: AWS IoT Greengrass

- Versiones 2.11.0, 2.10.0 y 2.9.5 del núcleo de Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan las versiones 2.11.0, 2.10.0 y 2.9.5 del software AWS IoT Greengrass Core.
- Suma compatibilidad para almacenar los valores de los datos de usuario de IDT en el almacén de parámetros de AWS Systems Manager y recuperarlos en la configuración mediante la sintaxis de marcadores de posición.
- Correcciones de errores menores.

Versión del conjunto de pruebas:

`GGV2Q_2.5.0`

- Publicado el 13 de diciembre de 2022

IDT v4.5.11 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan las versiones 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 y 2.6.0 del software AWS IoT Greengrass Core.
- Añade compatibilidad para probar PreInstalled Greengrass en un dispositivo central.
- Correcciones de errores menores.

Versión del conjunto de pruebas:

`GGV2Q_2.4.1`

- Publicado el 13 de octubre de 2022

IDT v4.5.8 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan las versiones 2.7.0, 2.6.0 y 2.5.6 del software AWS IoT Greengrass Core.
- Le permite realizar pruebas con PreInstalled Greengrass en un dispositivo central.
- Correcciones de errores menores.

Versión del conjunto de pruebas:

`GGV2Q_2.4.0`

- Publicado el 12 de agosto de 2022

IDT v4.5.3 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan las versiones 2.7.0, 2.6.0, 2.5.6, 2.5.5, 2.5.4 y 2.5.3 del software AWS IoT Greengrass Core.
- Actualiza la DockerApplicationManager prueba para usar una imagen de docker basada en ECR.
- Correcciones de errores menores.

Versión del conjunto de pruebas:

`GGV2Q_2.3.1`

- Publicado el 15 de abril de 2022

IDT v4.5.1 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan la versión 2.5.3 del software AWS IoT Greengrass Core.
- Suma compatibilidad para validar y calificar los dispositivos basados en Linux que utilizan un módulo de seguridad de hardware (HSM) para almacenar la clave privada y el certificado que utiliza el software AWS IoT Greengrass Core.
- Implementa el nuevo orquestador de pruebas de IDT para configurar conjuntos de pruebas personalizados. Para obtener más información, consulte Configuración del orquestador de pruebas de IDT.

- Correcciones de errores menores adicionales.

Versión del conjunto de pruebas:

`GGV2Q_2.3.0`

- Publicado el 11 de enero de 2022

IDT v4.4.1 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan la versión 2.5.2 del software AWS IoT Greengrass Core.
- Añade compatibilidad con el uso de una función de IAM definida por el usuario como función de intercambio de fichas que el dispositivo objeto de prueba asume al interactuar con los recursos. AWS

  Puede especificar el rol de IAM en el archivo `userdata.json`. Si especifica un rol personalizado, IDT utilizará ese rol en lugar de crear el rol de intercambio de tokens predeterminado durante la ejecución de la prueba.
- Correcciones de errores menores adicionales.

Versión del conjunto de pruebas:

`GGV2Q_2.2.1`

- Publicado el 12 de diciembre de 2021

IDT v4.4.0 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan la versión 2.5.0 del software AWS IoT Greengrass Core.
- Añade soporte para validar y calificar los dispositivos que ejecutan el software AWS IoT Greengrass Core en Windows.
- Admite el uso de la validación de claves públicas para las conexiones de dispositivos Secure Shell (SSH).
- Mejora la política de IAM de permisos de IDT con las prácticas recomendadas de seguridad.
- Correcciones de errores menores adicionales.

Versión del conjunto de pruebas:

`GGV2Q_2.1.0`

- Publicado el 19 de noviembre de 2021

IDT v4.2.0 para AWS IoT Greengrass

Notas de la versión:

- Incluye soporte para la calificación de las siguientes funciones en dispositivos que ejecutan el software AWS IoT Greengrass Core v2.2.0 y versiones posteriores:
  - Docker: valida que los dispositivos puedan descargar una imagen de contenedor de Docker de Amazon Elastic Container Registry (Amazon ECR).
  - Aprendizaje automático: valida que los dispositivos puedan realizar inferencias de aprendizaje automático (ML) mediante los marcos Deep Learning Runtime o Lite ML. TensorFlow
  - Stream Manager: valida que los dispositivos puedan descargar, instalar y ejecutar el administrador de transmisiones. AWS IoT Greengrass
- Le permite validar y calificar los dispositivos que ejecutan la versión 2.4.0, 2.3.0, 2.2.0 y 2.1.0 del software AWS IoT Greengrass Core.
- Agrupa los registros de prueba de cada caso de prueba en una *<test-case-id>* carpeta independiente dentro del directorio. *<device-tester-extract-location>*/`results/`*<execution-id>*`/logs/`*<test-group-id>*
- Correcciones de errores menores adicionales.

Versión del conjunto de pruebas:

`GGV2Q_2.0.1`

- Publicado el 31 de agosto de 2021

IDT v4.1.0 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan la versión 2.3.0, 2.2.0, 2.1.0 y 2.0.5 del software AWS IoT Greengrass Core.
- Mejora la configuración `userdata.json` al eliminar el requisito de especificar las propiedades `GreengrassNucleusVersion` y `GreengrassCLIVersion`.
- Incluye compatibilidad con la calificación de funciones Lambda y MQTT para el software AWS IoT Greengrass Core v2.1.0 y versiones posteriores. Ahora puede usar IDT para AWS IoT Greengrass V2 para validar que su dispositivo principal puede ejecutar funciones de Lambda y que el dispositivo puede publicar temas de MQTT y suscribirse AWS IoT Core a ellos.
- Mejora las capacidades de registro.

- Correcciones de errores menores adicionales.

Versión del conjunto de pruebas:

`GGV2Q_1.1.1`

- Publicado el 18 de junio de 2021

IDT v4.0.2 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan la versión 2.1.0 del software AWS IoT Greengrass Core.
- Añade compatibilidad con la calificación de funciones Lambda y MQTT para el software AWS IoT Greengrass Core v2.1.0 y versiones posteriores. Ahora puede usar IDT para AWS IoT Greengrass V2 para validar que su dispositivo principal puede ejecutar funciones de Lambda y que el dispositivo puede publicar temas de MQTT y suscribirse AWS IoT Core a ellos.
- Mejora las capacidades de registro.
- Correcciones de errores menores adicionales.

Versión del conjunto de pruebas:

`GGV2Q_1.1.1`

- Publicado el 05 de mayo de 2021

IDT v4.0.1 para AWS IoT Greengrass

Notas de la versión:

- Le permite validar y calificar los dispositivos que ejecutan la versión 2 del software de AWS IoT Greengrass .
- Le permite desarrollar y ejecutar sus conjuntos de pruebas personalizados utilizando for. AWS IoT Device Tester AWS IoT Greengrass Para obtener más información, consulte Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas.
- Proporciona aplicaciones IDT con firma de código para macOS y Windows. En macOS, es posible que tenga que conceder una excepción de seguridad para IDT. Para obtener más información, consulte Excepción de seguridad en macOS.

Versión del conjunto de pruebas:

`GGV2Q_1.0.0`

- Publicado el 22 de diciembre de 2020

- El conjunto de pruebas solo ejecuta las pruebas obligatorias para la calificación, a menos que se establezca el `value` correspondiente en la matriz `features` en `yes`.

Descarga de IDT para AWS IoT Greengrass V2

En este tema, se describen las opciones para descargar AWS IoT Device Tester para AWS IoT Greengrass V2. Puede utilizar uno de los siguientes enlaces de descarga de software o seguir las instrucciones para descargar IDT mediante programación.

Temas

- Descarga de IDT manualmente
- Descarga de IDT mediante programación

Al descargar el software, acepta el Acuerdo de licencia de AWS IoT Device Tester.

> **Note**
>
> IDT no admite la ejecución por parte de varios usuarios desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.

Descarga de IDT manualmente

En este tema, se muestran las versiones compatibles de IDT para AWS IoT Greengrass V2. Como práctica recomendada, le recomendamos que utilice la versión más reciente de IDT para la versión 2 de AWS IoT Greengrass que sea compatible con la versión de destino de la versión 2 de AWS IoT Greengrass. Los nuevos lanzamientos de AWS IoT Greengrass podrían requerir la descarga de una nueva versión de IDT para la versión 2 de AWS IoT Greengrass. Recibirá una notificación cuando inicie una ejecución de prueba si IDT para AWS IoT Greengrass V2 no es compatible con la versión de AWS IoT Greengrass que está utilizando.

Versión 4.9.4 de IDT para AWS IoT Greengrass

Versiones de AWS IoT Greengrass compatibles:

- Versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 del núcleo de Greengrass

Descargas de software de IDT:

- Versión 4.9.4 de IDT con el conjunto de pruebas GGV2Q_2.5.4 para Linux
- Versión 4.9.4 de IDT con el conjunto de pruebas GGV2Q_2.5.4 para macOS
- Versión 4.9.4 de IDT con el conjunto de pruebas GGV2Q_2.5.4 para Windows

Notas de la versión:

- Permite la validación y calificación de dispositivos que ejecutan las versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 del software AWS IoT Greengrass Core.
- Elimina los grupos de pruebas del administrador de flujos y machine learning.

Notas adicionales:

- Si su dispositivo usa un HSM y usted usa un núcleo 2.10.x, migre a la versión 2.11.0 o posterior del núcleo de Greengrass.

Versión del conjunto de pruebas:

`GGV2Q_2.5.4`

- Publicado el 03 de mayo de 2024

Descarga de IDT mediante programación

IDT proporciona una operación de API que puede utilizar para recuperar una URL desde la que descargar IDT mediante programación. También puede usar esta operación de API para comprobar si tiene la última versión de IDT. Esta operación de API tiene el siguiente punto de conexión.

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

Para llamar a esta operación de API, debe tener el permiso para realizar la acción **`iot-device-tester:LatestIdt`**. Incluya su firma de AWS y use `iot-device-tester` como nombre del servicio.

Solicitud de API

HostOS: el sistema operativo de la máquina host. Puede elegir entre las siguientes opciones:

- `mac`
- `linux`
- `windows`

TestSuiteType: el tipo de conjunto de pruebas. Elija la opción siguiente:

`GGV2`: IDT para AWS IoT Greengrass V2

Versión del producto

(Opcional) La versión del núcleo de Greengrass. El servicio devuelve la última versión compatible de IDT para esa versión del núcleo de Greengrass. Si no especifica esta opción, el servicio devuelve la última versión de IDT.

Respuesta de la API

La respuesta de la API tiene el siguiente formato. `DownloadURL` incluye un archivo zip.

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

Ejemplos

Puede hacer referencia a los siguientes ejemplos para descargar IDT mediante programación. En estos ejemplos se utilizan las credenciales que almacena en las variables de entorno `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY`. Para seguir las mejores prácticas recomendadas, no almacene las credenciales en el código.

Example Ejemplo: descarga con cURL 7.75.0 o posterior (Mac y Linux)

Si tiene la versión 7.75.0 o posterior de cURL, puede usar el indicador `aws-sigv4` para firmar la solicitud de API. En este ejemplo se usa jq para analizar la URL de descarga de la respuesta.

> **Warning**
>
> El indicador `aws-sigv4` requiere que los parámetros de consulta de la solicitud CURL GET estén en el orden de HostOs/ProductVersion/TestSuiteType o HostOs/TestSuiteType. Los

órdenes que no se ajusten, provocarán un error al obtener firmas no coincidentes para la cadena canónica de la puerta de enlace de la API.
Si se incluye el parámetro opcional ProductVersion, debe usar una versión de producto compatible, tal como se describe en Versiones compatibles de AWS IoT Device Tester para AWS IoT Greengrass V2.

- Sustituya *us-west-2* por su Región de AWS. Para obtener la lista de códigos de región, consulte Puntos de conexión regionales.
- Sustituya *linux* por el sistema operativo de su máquina host.
- Sustituya *2.5.3* por su versión del núcleo de AWS IoT Greengrass.

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example Ejemplo: descarga con una versión anterior de cURL (Mac y Linux)

Puede usar el siguiente comando cURL con una firma de AWS que firme y calcule. Para obtener más información sobre cómo firmar y calcular una firma de AWS, consulte Firma de solicitudes de API de AWS.

- Sustituya *linux* por el sistema operativo de su máquina host.
- Sustituya *Timestamp* por la fecha y la hora, por ejemplo, **20220210T004606Z**.
- Sustituya *Date* por la fecha, por ejemplo, **20220210**.
- Sustituya *AWSRegion* por su Región de AWS. Para obtener la lista de códigos de región, consulte Puntos de conexión regionales.
- Sustituya *AWSSignature* por la firma de AWS que haya generado.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example Ejemplo: descarga mediante un script de Python

En este ejemplo se utiliza la biblioteca de solicitudes de Python. Este ejemplo está adaptado del ejemplo de Python para firmar una solicitud de API de AWS en la Referencia general de AWS.

- Sustituya *us-west-2* por su región. Para obtener la lista de códigos de región, consulte Puntos de conexión regionales.
- Sustituya *linux* por el sistema operativo de su máquina host.

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'
```

```
# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
```

```
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
```

```
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

Utilice IDT para ejecutar el conjunto de AWS IoT Greengrass cualificaciones

Puede utilizar la AWS IoT Greengrass versión 2 AWS IoT Device Tester para comprobar que el software AWS IoT Greengrass principal se ejecuta en su hardware y se puede comunicar con ella. Nube de AWS También realiza end-to-end pruebas con AWS IoT Core. Por ejemplo, verifica que su dispositivo pueda implementar componentes y actualizarlos.

Además de probar los dispositivos, IDT for AWS IoT Greengrass V2 crea recursos (por ejemplo, AWS IoT cosas, grupos, etc.) Cuenta de AWS para facilitar el proceso de calificación.

Para crear estos recursos, IDT for AWS IoT Greengrass V2 utiliza las AWS credenciales configuradas en el `config.json` archivo para realizar llamadas a la API en su nombre. Estos recursos se aprovisionarán en distintos momentos durante una prueba.

Cuando se utiliza IDT para AWS IoT Greengrass V2 para ejecutar el paquete de AWS IoT Greengrass cualificación, éste lleva a cabo los siguientes pasos:

1. Carga y valida su dispositivo y la configuración de credenciales.
2. Realiza pruebas seleccionadas con los recursos locales y de la nube necesarios.
3. Depura los recursos locales y de la nube.
4. Genera informes de pruebas que indican si la placa supera las pruebas necesarias para la cualificación.

Versiones del conjunto de pruebas

IDT para AWS IoT Greengrass V2 organiza las pruebas en conjuntos de pruebas y grupos de pruebas.

- Un conjunto de pruebas es el conjunto de grupos de pruebas que se utiliza para verificar que un dispositivo funciona con versiones particulares de AWS IoT Greengrass.
- Un grupo de pruebas es el conjunto de pruebas individuales relacionadas con una característica concreta, como implementaciones de componentes.

Los conjuntos de pruebas se versionan mediante un formato *major*.*minor*.*patch*, por ejemplo, `GGV2Q_1.0.0`. Al descargar IDT, el paquete incluye la versión más reciente del conjunto de calificación de Greengrass.

> **Important**
>
> Las pruebas de versiones del conjunto de pruebas no compatibles no son válidas para la cualificación del dispositivo. IDT no imprime informes de cualificación para versiones no compatibles. Para obtener más información, consulte the section called “Política de soporte de AWS IoT Device Tester para AWS IoT Greengrass”.
> Puede ejecutar `list-supported-products` una lista de las versiones AWS IoT Greengrass y los conjuntos de pruebas compatibles con su versión actual de IDT.

Descripciones de los grupos de pruebas

Grupos de pruebas necesarias para la cualificación del núcleo

Estos grupos de prueba son necesarios para que su dispositivo AWS IoT Greengrass V2 pueda incluirse en el catálogo de AWS Partner dispositivos.

Dependencias de Core

Valida que el dispositivo cumpla todos los requisitos de software y hardware del software AWS IoT Greengrass Core. Este grupo de pruebas incluye el siguiente caso de prueba:

Versión de Java

Comprueba que la versión de Java requerida esté instalada en el dispositivo que se está probando. AWS IoT Greengrass requiere Java 8 o una versión posterior.

PreTest Validación

Comprueba que el dispositivo cumple los requisitos de software para ejecutar las pruebas.

- En el caso de los dispositivos basados en Linux, esta prueba comprueba si el dispositivo puede ejecutar los siguientes comandos de Linux:

`chmod`, `cp`, `echo`, `grep`, `kill`, `ln`, `mkinfo`, `ps`, `rm`, `sh`, `uname`

- En el caso de los dispositivos basados en Windows, esta prueba comprueba si el dispositivo tiene instalado el siguiente software de Microsoft:

  Utilidad Powershell v5.1 o posterior, .NET v4.6.1 o posterior, Visual C++ 2017 o posterior PsExec

Comprobador de versiones

Comprueba que la versión AWS IoT Greengrass proporcionada es compatible con la versión de AWS IoT Device Tester que esté utilizando.

Componente

Valida que el dispositivo pueda implementar componentes y actualizarlos. Este grupo de pruebas incluye las siguientes pruebas:

Componente en la nube

Valida la capacidad del dispositivo para los componentes en la nube.

Componente local

Valida la capacidad del dispositivo para los componentes locales.

Lambda

Esta prueba no se aplica a los dispositivos basados en Windows.

Valida que el dispositivo pueda implementar componentes de funciones de Lambda que usen el tiempo de ejecución de Java y que las funciones de Lambda puedan AWS IoT Core usar temas de MQTT como fuentes de eventos para los mensajes de trabajo.

MQTT

Valida que el dispositivo pueda suscribirse a temas de MQTT y publicarlos en ellos. AWS IoT Core

Grupos de pruebas opcionales

> **Note**
>
> Estos grupos de prueba son opcionales y se utilizan únicamente para los dispositivos principales de Greengrass basados en Linux que reúnan los requisitos. Si decide optar

a las pruebas opcionales, su dispositivo aparece con capacidades adicionales en el catálogo de AWS Partner dispositivos.

Dependencias de Docker

Valida que el dispositivo cumpla con todas las dependencias técnicas necesarias para utilizar el componente Docker Application Manager () AWS suministrado. `aws.greengrass.DockerApplicationManager`

Calificación del administrador de aplicaciones de Docker

Valida que el dispositivo pueda descargar una imagen de contenedor de Docker desde Amazon ECR.

Dependencias de machine learning

> **Note**
>
> El grupo de prueba opcional de machine learning solo es compatible con la versión 4.9.3 de IDT.

Valida que el dispositivo cumpla con todas las dependencias técnicas necesarias para utilizar los componentes de aprendizaje automático AWS(ML) proporcionados.

Pruebas de inferencia de machine learning

> **Note**
>
> El grupo de prueba opcional de machine learning solo es compatible con la versión 4.9.3 de IDT.

Valida que el dispositivo pueda realizar inferencias de aprendizaje automático mediante los marcos Deep Learning Runtime y Lite ML. TensorFlow

Dependencias del administrador de flujos

> **Note**
>
> El grupo de prueba opcional del administrador de flujos solo es compatible con la versión 4.9.3 de IDT.

Valida que los dispositivos puedan descargar, instalar y ejecutar el administrador de flujos de AWS IoT Greengrass.

Integración de la seguridad por hardware (HSI)

> **Note**
>
> Esta prueba está disponible en la versión 4.9.3 de IDT y versiones posteriores únicamente para dispositivos basados en Linux. AWS IoT Greengrass actualmente no admite la integración de seguridad de hardware para dispositivos Windows.

Valida que el dispositivo pueda autenticar las conexiones a los AWS IoT AWS IoT Greengrass servicios mediante una clave privada y un certificado almacenados en un módulo de seguridad de hardware (HSM). Esta prueba también verifica que el componente del proveedor PKCS #11 AWS proporcionado pueda interactuar con el HSM mediante una biblioteca PKCS #11 proporcionada por el proveedor. Para obtener más información, consulte Integración de la seguridad de hardware.

Requisitos previos para ejecutar el paquete de AWS IoT Greengrass calificación

En esta sección se describen los requisitos previos para usar AWS IoT Device Tester (IDT) para. AWS IoT Greengrass

Descargue la última versión de para AWS IoT Device Tester AWS IoT Greengrass

Descargue la última versión de IDT y extraiga el software en una ubicación (*<device-tester-extract-location>*) del sistema de archivos en la que tenga permisos de lectura/escritura.

> **Note**
>
> IDT no admite la ejecución por parte de varios usuarios desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si utiliza Windows, extraiga IDT en un directorio raíz como `C:\` o `D:\` para mantener las rutas por debajo del límite de 260 caracteres.

Descargue el software AWS IoT Greengrass

IDT for AWS IoT Greengrass V2 comprueba la compatibilidad de su dispositivo con una versión específica de AWS IoT Greengrass. Ejecute el siguiente comando para descargar el software AWS IoT Greengrass principal a un archivo denominado`aws.greengrass.nucleus.zip`. *`version`*Sustitúyala por una versión de componentes Nucleus compatible para su versión de IDT.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip -
OutFile aws.greengrass.nucleus.zip
```

Coloque el archivo descargado `aws.greengrass.nucleus.zip` en la carpeta *`<device-tester-extract-location>`*`/products/`.

> **Note**
>
> No coloque varios archivos en este directorio para el mismo sistema operativo y arquitectura.

Cree y configure un Cuenta de AWS

Antes de poder utilizarla AWS IoT Device Tester para la AWS IoT Greengrass V2, debe realizar los siguientes pasos:

1. Configure un Cuenta de AWS. Si ya tiene una Cuenta de AWS, vaya al paso 2.
2. Configurar permisos de IDT.

Estos permisos de cuenta permiten a IDT acceder a los AWS servicios y crear AWS recursos, como AWS IoT cosas y AWS IoT Greengrass componentes, en su nombre.

Para crear estos recursos, IDT para AWS IoT Greengrass V2 utiliza las AWS credenciales configuradas en el `config.json` archivo para realizar llamadas a la API en su nombre. Estos recursos se aprovisionarán en distintos momentos durante una prueba.

> **Note**
>
> Aunque la mayoría de las pruebas cumplen los requisitos para el Nivel gratuito de AWS, debe proporcionar una tarjeta de crédito cuando se cree una Cuenta de AWS. Para obtener más información, consulte ¿Por qué necesito un método de pago si mi cuenta está cubierta por la capa gratuita?.

Paso 1: configurar una Cuenta de AWS

En este paso, cree y configure una Cuenta de AWS. Si ya tiene una Cuenta de AWS, vaya directamente a the section called “Paso 2: Configurar los permisos de IDT”.

Si no tiene uno Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abrir https://portal.aws.amazon.com/billing/registro.

2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

   Cuando te registras en un Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar Tareas que requieren acceso de usuario raíz.

Para crear un usuario administrador, elija una de las siguientes opciones.

Elegir una forma de administrar el administrador	Para	Haga esto	También puede
En IAM Identity Center (recomendado)	Usar credenciales a corto plazo para acceder a AWS. Esto se ajusta a las prácticas recomendadas de seguridad. Para obtener información sobre las prácticas recomendadas, consulta Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.	Siga las instrucciones en Introducción en la Guía del usuario de AWS IAM Identity Center .	Configure el acceso programático configurando el AWS CLI que se utilizará AWS IAM Identity Center en la Guía del AWS Command Line Interface usuario.

Elegir una forma de administrar el administrador	Para	Haga esto	También puede
En IAM (no recomendado)	Usar credenciales a largo plazo para acceder a AWS.	Siguiendo las instrucciones de Crear un usuario de IAM para acceso de emergencia de la Guía del usuario de IAM.	Configure el acceso programático mediante Administrar las claves de acceso de los usuarios de IAM en la Guía del usuario de IAM.

Paso 2: Configurar los permisos de IDT

En este paso, configure los permisos que IDT for AWS IoT Greengrass V2 utiliza para ejecutar pruebas y recopilar datos de uso de IDT. Puede utilizar la Consola de administración de AWS o la AWS Command Line Interface (AWS CLI) para crear una política de IAM y un usuario de prueba para IDT y, a continuación, asociar políticas al usuario. Si ya ha creado un usuario de prueba para IDT, vaya a Configuración de su dispositivo para ejecutar pruebas de IDT.

Configuración de permisos de IDT (consola)

1. Inicie sesión en la consola de IAM.
2. Crear una política administrada que conceda permisos para crear roles con permisos específicos.

   a. En el panel de navegación, seleccione Políticas y, a continuación, Crear política.

   b. Si no los está utilizando PreInstalled, en la pestaña JSON, sustituya el contenido del marcador de posición por la siguiente política. Si lo está utilizando PreInstalled, continúe con el siguiente paso.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
```

```
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
```

```
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
```

```
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
```

```
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

c. Si lo está utilizando PreInstalled, en la pestaña JSON, sustituya el contenido del marcador de posición por la siguiente política. Asegúrese de hacer lo siguiente:

- Sustituya *`thingName`* y *`thingGroup`* en la `iotResources` instrucción por el nombre y el grupo de cosas que se crearon durante la instalación de Greengrass en el dispositivo que se está probando (DUT) para añadir permisos.
- Sustituya las *`roleAlias`* letras *`passRole`* y en la `roleAliasResources` declaración y la `passRoleForResources` declaración por las funciones que se crearon durante la instalación de Greengrass en su DUT.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
```

```
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
```

```
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
```

```
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
```

```
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> Si desea utilizar un rol de IAM personalizado como rol de intercambio de token para el dispositivo que se está probando, asegúrese de actualizar la declaración `roleAliasResources` y la declaración `passRoleForResources` de su política para permitir el recurso de su rol de IAM personalizado.

d. Elija Revisar política.

e. En Nombre, ingrese **`IDTGreengrassIAMPermissions`**. En Summary (Resumen), revise los permisos concedidos por la política.

f. Elija Crear política.

3. Cree un usuario de IAM y adjunte los permisos requeridos por IDT para AWS IoT Greengrass.

   a. Cree un usuario de IAM. Siga los pasos del 1 al 5 en Creación de usuarios de IAM (consola) en la Guía del usuario de IAM.

   b. Adjunte los permisos a su usuario de IAM:

      i. En la página Set permissions (Establecer permisos), elija Attach existing policies to user directly (Adjuntar políticas existentes al usuario directamente).

      ii. Busque la IDTGreengrassIAMPermissionspolítica que creó en el paso anterior. Seleccione la casilla de verificación.

   c. Elija Siguiente: etiquetas.

   d. Elija Next: Review (Siguiente: revisar) para ver un resumen de sus opciones.

   e. Seleccione la opción Crear un usuario.

   f. Para ver las claves de acceso del usuario (clave de acceso IDs y claves de acceso secretas), selecciona Mostrar junto a la contraseña y la clave de acceso. Para guardar las claves de acceso, elija Download.csv (Descargar archivo .csv) y, a continuación, guarde el

archivo en un lugar seguro. Utilice esta información más adelante para configurar su archivo de credenciales de AWS .

4. Siguiente paso: Configure su dispositivo físico.

Configuración de permisos de IDT (AWS CLI)

1. En su ordenador, instale y configure la AWS CLI opción si aún no está instalada. Siga los pasos que se indican en Instalación de la AWS CLI en la Guía del usuario de la AWS Command Line Interface .

> **Note**
>
> AWS CLI Se trata de una herramienta de código abierto que puede utilizar para interactuar con los AWS servicios desde el shell de la línea de comandos.

2. Cree una política administrada por el cliente que conceda permisos para administrar IDT y roles de AWS IoT Greengrass .

   a. Si no la está utilizando PreInstalled, abra un editor de texto y guarde el siguiente contenido de la política en un archivo JSON. Si lo está utilizando PreInstalled, continúe con el siguiente paso.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
```

```
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
```

```
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
```

```
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
```

```
    }
  ]
}
```

b. Si lo está utilizando PreInstalled, abra un editor de texto y guarde el siguiente contenido de la política en un archivo JSON. Asegúrese de hacer lo siguiente:

- Sustituya *thingName* y *thingGroup* en la `iotResources` declaración que se creó durante la instalación de Greengrass en su dispositivo bajo prueba (DUT) para añadir permisos.
- Sustituya las *roleAlias* letras *passRole* y en la `roleAliasResources` declaración y la `passRoleForResources` declaración por las funciones que se crearon durante la instalación de Greengrass en su DUT.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
```

```
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
```

```
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
```

```
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

Note

Si desea utilizar un rol de IAM personalizado como rol de intercambio de token para el dispositivo que se está probando, asegúrese de actualizar la declaración

`roleAliasResources` y la declaración `passRoleForResources` de su política para permitir el recurso de su rol de IAM personalizado.

c. Ejecute el siguiente comando para crear una política administrada por el cliente denominada `IDTGreengrassIAMPermissions`. Reemplace *`policy.json`* por la ruta completa del archivo JSON que creó en el paso anterior.

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document file://policy.json
```

3. Cree un usuario de IAM y adjunte los permisos requeridos por IDT para AWS IoT Greengrass.

a. Cree un usuario de IAM. En esta configuración de ejemplo, el usuario se denomina `IDTGreengrassUser`.

```
aws iam create-user --user-name IDTGreengrassUser
```

b. Asocie la política `IDTGreengrassIAMPermissions` que creó en el paso 2 a su usuario de IAM. Sustituya *`<account-id>`* el comando por el ID de su. Cuenta de AWS

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
 arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. Cree una clave de acceso secreta para el usuario.

```
aws iam create-access-key --user-name IDTGreengrassUser
```

Almacene la salida en una ubicación segura. Utilizará esta información más adelante para configurar el archivo de AWS credenciales.

5. Siguiente paso: Configure su dispositivo físico.

AWS IoT Device Tester permisos

Las siguientes políticas describen AWS IoT Device Tester los permisos.

AWS IoT Device Tester requiere estos permisos para las funciones de comprobación de versiones y actualización automática.

- `iot-device-tester:SupportedVersion`

Otorga AWS IoT Device Tester permiso para obtener la lista de productos, conjuntos de pruebas y versiones de IDT compatibles.

- `iot-device-tester:LatestIdt`

  Otorga AWS IoT Device Tester permiso para obtener la última versión de IDT disponible para su descarga.

- `iot-device-tester:CheckVersion`

  Otorga AWS IoT Device Tester permiso para comprobar la compatibilidad de las versiones de IDT, los conjuntos de pruebas y los productos.

- `iot-device-tester:DownloadTestSuite`

  Otorga AWS IoT Device Tester permiso para descargar las actualizaciones de los conjuntos de pruebas.

AWS IoT Device Tester también utiliza el siguiente permiso para la elaboración de informes de métricas opcionales:

- `iot-device-tester:SendMetrics`

  Otorga permiso AWS para recopilar métricas sobre el uso AWS IoT Device Tester interno. Si se omite este permiso, no se recopilarán estas métricas.

Configuración de su dispositivo para ejecutar pruebas de IDT

Para permitir que IDT ejecute pruebas de calificación del dispositivo, debe configurar su computadora host para acceder a su dispositivo y configurar los permisos de usuario en su dispositivo.

Instalación de Java en la computadora host

A partir de la versión 4.2.0 de IDT, las pruebas de calificación opcionales AWS IoT Greengrass requieren la ejecución de Java.

Puede utilizar la versión 8 o posterior de Java. Le recomendamos que utilice las versiones de compatibilidad a largo plazo de Amazon Corretto u OpenJDK. Se requiere la versión 8 o posterior.

Configuración del equipo host para acceder a un dispositivo en pruebas

IDT se ejecuta en su equipo host y debe poder utilizar SSH para conectarse a su dispositivo. Existen dos opciones para permitir que IDT obtenga acceso SSH a los dispositivos sometidos a la prueba:

1. Siga las instrucciones que se indican aquí para crear un par de claves SSH y autorizar su clave para iniciar sesión en su dispositivo en proceso de prueba sin especificar una contraseña.
2. Proporcione un nombre de usuario y una contraseña para cada dispositivo en el archivo `device.json`. Para obtener más información, consulte Configurar device.json.

Puede utilizar cualquier implementación SSL para crear una clave SSH. Las siguientes instrucciones muestran cómo utilizar SSH-KEYGEN o Pu (para Windows). TTYgen Si utiliza otra implementación de SSL, consulte la documentación de dicha aplicación.

IDT utiliza claves SSH para autenticar con su dispositivo bajo prueba.

Para crear una clave SSH con SSH-KEYGEN, realice el siguiente procedimiento:

1. Cree una clave de SSH.

   Puede utilizar el comando ssh-keygen de Open SSH para crear un par de claves SSH. Si ya tiene un par de claves SSH en su equipo host, es una práctica recomendada crear un par de claves SSH específicamente para IDT. De esta forma, una vez completadas las pruebas, el equipo host ya no podrá conectarse a su dispositivo sin introducir una contraseña. También le permite restringir el acceso al dispositivo remoto únicamente a aquellos que lo necesiten.

   > **Note**
   >
   > Windows no tiene instalado un cliente SSH. Para obtener más información sobre la instalación de un cliente SSH en Windows, consulte Download SSH Client Software.

   El comando ssh-keygen le solicita un nombre y la ruta para almacenar el par de claves. De forma predeterminada, los archivos de par de claves se denominan `id_rsa` (clave privada) y `id_rsa.pub` (clave pública). En macOS y Linux, la ubicación predeterminada de estos archivos es `~/.ssh/`. En Windows, la ubicación predeterminada es `C:\Users\<user-name>\.ssh`.

   Cuando se le solicite, introduzca una frase clave para proteger la clave SSH. Para obtener más información, consulte la sección acerca de cómo generar una nueva clave SSH.

2. Añada claves SSH autorizadas a su dispositivo en proceso de prueba.

   IDT debe utilizar su clave privada de SSH para iniciar sesión en el dispositivo a prueba. Para autorizar que su clave privada de SSH inicie sesión en el dispositivo a prueba, use el comando ssh-copy-id en su equipo host. Este comando añade su clave pública al archivo `~/.ssh/authorized_keys` que se encuentra en su dispositivo a prueba. Por ejemplo:

   $ ssh-copy-id *<remote-ssh-user>*@*<remote-device-ip>*

   ¿Dónde *remote-ssh-user* está el nombre de usuario que se utiliza para iniciar sesión en el dispositivo que *remote-device-ip* se está probando y la dirección IP del dispositivo que se está probando? Por ejemplo:

   ssh-copy-id pi@192.168.1.5

   Cuando se le solicite, introduzca la contraseña para el nombre de usuario que ha especificado en el comando ssh-copy-id.

   ssh-copy-id presupone que la clave pública se denomina `id_rsa.pub` y se almacena en la ubicación predeterminada (en macOS y Linux, `~/.ssh/` y en Windows, `C:\Users\`*<user-name>*`\.ssh`). Si asignó a la clave pública un nombre diferente o la almacenó en otra ubicación, debe especificar la ruta completa a su clave pública SSH utilizando la opción -i para ssh-copy-id (por ejemplo: ssh-copy-id -i ~/my/path/myKey.pub). Para obtener más información acerca de la creación de claves de SSH y la copia de las claves públicas, consulte SSH-COPY-ID.

Para crear una clave SSH con Pu TTYgen (solo para Windows)

1. Asegúrese de que tiene el servidor y el cliente de OpenSSH instalados en su dispositivo en proceso de prueba. Para obtener más información, consulte OpenSSH.
2. Instala Pu TTYgen en el dispositivo que estés probando.
3. Abre PuTTYgen.
4. Elija Generate (Generar) y mueva el cursor del ratón dentro del cuadro para generar una clave privada.
5. En el menú Conversions (Conversiones), elija Export OpenSSH key (Exportar clave OpenSSH) y guarde la clave privada con una extensión de archivo `.pem`.
6. Añada la clave pública al archivo `/home/`*<user>*`/.ssh/authorized_keys` en el dispositivo en proceso de prueba.

a. Copia el texto de la clave pública de la TTYgen ventana Pu.

b. Utilice PuTTY para crear una sesión en su dispositivo en proceso de prueba.

   i. En un símbolo del sistema o en una ventana de Windows PowerShell, ejecute el siguiente comando:

      C:/*<path-to-putty>*/putty.exe -ssh *<user>*@*<dut-ip-address>*

   ii. Cuando se le solicite, escriba la contraseña de su dispositivo.

   iii. Utilice vi u otro editor de texto para añadir la clave pública al archivo `/home/<user>/.ssh/authorized_keys` en su dispositivo en proceso de prueba.

7. Actualice el archivo `device.json` con su nombre de usuario, la dirección IP y la ruta al archivo de clave privada que acaba de guardar en el equipo host para cada dispositivo en proceso de prueba. Para obtener más información, consulte the section called “Configurar device.json”. Asegúrese de proporcionar la ruta completa y el nombre de archivo a la clave privada y utilizar barras diagonales (“/”). Por ejemplo, para la ruta de Windows `C:\DT\privatekey.pem`, utilice `C:/DT/privatekey.pem` en el archivo `device.json`.

Configuración de las credenciales de usuario para los dispositivos Windows

Para calificar un dispositivo basado en Windows, debe configurar las credenciales de usuario en la LocalSystem cuenta del dispositivo que se está probando para los siguientes usuarios:

- El usuario predeterminado de Greengrass (`ggc_user`).
- El usuario que utiliza para conectarse al dispositivo que se está probando. Este usuario se configura en el archivo `device.json`.

Debe crear cada usuario en la LocalSystem cuenta del dispositivo que se está probando y, a continuación, almacenar el nombre de usuario y la contraseña del usuario en la instancia de Credential Manager de la LocalSystem cuenta.

Configuración de los usuarios en dispositivos Windows

1. Abra el símbolo del sistema de Windows (`cmd.exe`) como administrador.
2. Cree los usuarios en la LocalSystem cuenta del dispositivo Windows. Ejecute el siguiente comando para cada usuario que desee crear. Para el usuario predeterminado de Greengrass, sustitúyalo por*user-name*. `ggc_user` *password*Sustitúyala por una contraseña segura.

```
net user /add user-name password
```

3. Descargue e instale la PsExecutilidad de Microsoft en el dispositivo.
4. Utilice la PsExec utilidad para almacenar el nombre de usuario y la contraseña del usuario predeterminado en la instancia de Credential Manager de la LocalSystem cuenta.

   Ejecute el siguiente comando para cada usuario que desee configurar en el administrador de credenciales. Para el usuario predeterminado de Greengrass, sustitúyalo por*user-name*. `ggc_user` *password*Sustitúyala por la contraseña del usuario que configuraste anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:user-name /user:user-name /pass:password
   ```

   Si PsExec License Agreement se abre, elija Accept para aceptar la licencia y ejecute el comando.

   > **Note**
   >
   > En los dispositivos Windows, la LocalSystem cuenta ejecuta el núcleo de Greengrass y debe usar la PsExec utilidad para almacenar la información del usuario en la LocalSystem cuenta. El uso de la aplicación Credential Manager almacena esta información en la cuenta de Windows del usuario que ha iniciado sesión actualmente, en lugar de en la LocalSystem cuenta.

Configuración de los permisos de usuario en el dispositivo

IDT realiza operaciones en diversos directorios y archivos en un dispositivo que se está probando. Algunas de estas operaciones requieren permisos elevados (usando sudo). Para automatizar estas operaciones, IDT para AWS IoT Greengrass V2 debe poder ejecutar comandos con sudo sin que se le pida una contraseña.

Siga estos pasos en el dispositivo a prueba para permitir acceso a sudo sin que se le solicite una contraseña.

> **Note**
>
> `username` hace referencia al usuario de SSH que utiliza IDT para obtener acceso al dispositivo bajo prueba.

Para añadir el usuario al grupo sudo

1. En el dispositivo bajo prueba, ejecute `sudo usermod -aG sudo` *`<username>`*.
2. Cierre la sesión y, a continuación, vuelva a iniciar sesión para que los cambios surtan efecto.
3. Para comprobar que su nombre de usuario se haya añadido correctamente, ejecute sudo echo test. Si no se le solicita una contraseña, el usuario se ha configurado correctamente.
4. Añada el archivo `/etc/sudoers` y agregue la siguiente línea al final del archivo:

   `<ssh-username>` `ALL=(ALL) NOPASSWD: ALL`

Configuración de un rol de intercambio de token personalizado

Puede optar por utilizar una función de IAM personalizada como función de intercambio de fichas que el dispositivo objeto de prueba asume para interactuar con los recursos. AWS Para obtener información sobre la creación de un rol de IAM, consulte Creación de roles de IAM en la Guía del usuario de IAM.

Debe cumplir con los siguientes requisitos para permitir que IDT utilice su rol de IAM personalizado. Le recomendamos encarecidamente que agregue a este rol solo las acciones de política mínimas requeridas.

- El archivo de configuración userdata.json debe actualizarse para establecer el parámetro `GreengrassV2TokenExchangeRole` en `true`.
- El rol de IAM personalizado debe configurarse con la siguiente política de confianza mínima:

  JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
```

```
            "Service":[
               "credentials.iot.amazonaws.com",
               "lambda.amazonaws.com",
               "sagemaker.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

- El rol de IAM personalizado debe configurarse con la siguiente política de permisos mínimos:

  JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- El nombre del rol de IAM personalizado debe coincidir con el recurso del rol de IAM que especifique en los permisos de IAM para el usuario de prueba. De forma predeterminada, la política de usuarios de prueba permite el acceso a los roles de IAM que tienen el prefijo `idt-` en sus nombres de rol. Si el nombre de su rol de IAM no usa este prefijo, agregue el recurso `arn:aws:iam::*:role/`*`custom-iam-role-name`* a la declaración `roleAliasResources` y la declaración `passRoleForResources` a su política de usuarios de prueba, como se muestra en los siguientes ejemplos:

  Example **`passRoleForResources`** instrucción

```
{
   "Sid":"passRoleForResources",
   "Effect":"Allow",
   "Action":"iam:PassRole",
   "Resource":"arn:aws:iam::*:role/custom-iam-role-name",
   "Condition":{
      "StringEquals":{
         "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
         ]
      }
   }
}
```

  Example **`roleAliasResources`** instrucción

```
{
   "Sid":"roleAliasResources",
   "Effect":"Allow",
   "Action":[
      "iot:CreateRoleAlias",
      "iot:DescribeRoleAlias",
      "iot:DeleteRoleAlias",
      "iot:TagResource",
      "iam:GetRole"
   ],
   "Resource":[
      "arn:aws:iot:*:*:rolealias/idt-*",
      "arn:aws:iam::*:role/custom-iam-role-name"
```

```
    ]
}
```

Configurar el dispositivo para probar características opcionales

En esta sección, se describen los requisitos del dispositivo para ejecutar pruebas IDT para las características opcionales de Docker y machine learning (ML). Las características de ML solo son compatibles con la versión 4.9.3 de IDT. Debe asegurarse de que su dispositivo cumpla estos requisitos solo si desea probar estas características. De lo contrario, siga en the section called "Configuración de los ajustes de IDT".

Temas

- Requisitos de calificación de Docker
- Requisitos de calificación de ML
- Requisitos de calificación de HSM

Requisitos de calificación de Docker

IDT for AWS IoT Greengrass V2 ofrece pruebas de calificación de Docker para validar que sus dispositivos puedan usar el componente de administrador de aplicaciones de Docker que AWS se proporciona para descargar imágenes de contenedores de Docker que pueden ejecutarse con componentes de contenedores de Docker personalizados. Para obtener más información acerca de la creación de componentes de Docker, consulte Ejecución de un contenedor de Docker.

Para ejecutar las pruebas de calificación de Docker, los dispositivos que se estén probando deben cumplir los siguientes requisitos para implementar el componente administrador de aplicaciones de Docker.

- Versión 1.9.1 o posterior de Docker Engine instalada en el dispositivo principal de Greengrass. Se ha comprobado que la versión 20.10 es la última versión que funciona con el software Core. AWS IoT Greengrass Debe instalar Docker directamente en el dispositivo principal antes de implementar componentes que ejecuten contenedores de Docker.
- El daemon de Docker se inició y se ejecutó en el dispositivo principal antes de implementar este componente.
- El usuario del sistema que ejecute un componente contenedor de Docker debe tener permisos de raíz o administrador, o bien debe configurar Docker para que se ejecute como un usuario no de raíz o no administrador.

- En los dispositivos Linux, debe agregar un usuario al grupo de `docker` para llamar comandos `docker` sin `sudo`.
- En los dispositivos Windows, puede agregar un usuario al grupo de `docker-users` para llamar comandos `docker` sin privilegios de administrador.

Linux or Unix

Para agregar `ggc_user`, o el usuario no raíz que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker`, ejecute el siguiente comando.

```
sudo usermod -aG docker ggc_user
```

Para obtener más información, consulte Administrar Docker como un usuario no raíz.

Windows Command Prompt (CMD)

Para agregar `ggc_user`, o el usuario que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker-users`, ejecute el siguiente comando como un administrador.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Para agregar `ggc_user`, o el usuario que utilice para ejecutar los componentes del contenedor de Docker, al grupo de `docker-users`, ejecute el siguiente comando como un administrador.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

Requisitos de calificación de ML

> **Note**
>
> La característica de machine learning solo es compatible con la versión 4.9.3 de IDT.

IDT for AWS IoT Greengrass V2 ofrece pruebas de aptitud para el aprendizaje automático con el fin de validar que sus dispositivos puedan utilizar los componentes AWS de aprendizaje automático proporcionados para realizar inferencias de aprendizaje automático de forma local mediante los marcos Deep Learning Runtime o TensorFlow Lite ML. Para obtener más información sobre cómo

ejecutar la inferencia de ML en dispositivos de Greengrass, consulte Cómo realizar la inferencia de machine learning.

Para ejecutar las pruebas de calificación de ML, los dispositivos que se estén probando deben cumplir los siguientes requisitos para implementar los componentes de machine learning.

- En los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 o Ubuntu 18.04, se instala en el dispositivo la versión 2.27 o posterior de la Biblioteca C GNU (glibc).
- En los dispositivos ARMv7L, como Raspberry Pi, las dependencias para OpenCV-Python están instaladas en el dispositivo. Ejecute el siguiente comando para instalar las dependencias.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Los dispositivos Raspberry Pi que ejecutan el sistema operativo Bullseye de Raspberry Pi deben cumplir los siguientes requisitos:
  - NumPy 1.22.4 o una versión posterior instalada en el dispositivo. Raspberry Pi OS Bullseye incluye una versión anterior de NumPy, por lo que puede ejecutar el siguiente comando para actualizar NumPy el dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - La pila de cámara antigua habilitada en el dispositivo. El sistema operativo Bullseye de Raspberry Pi incluye una nueva pila de cámara que está habilitada de forma predeterminada y no es compatible, por lo que debe activar la pila de cámara antigua.

    Cómo activar la pila de cámara antigua

    1. Ejecute el siguiente comando para abrir la herramienta de configuración de Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Seleccione Opciones de interfaz.
    3. Seleccione Cámara antigua para activar la pila de cámara antigua.
    4. Reinicie el Raspberry Pi.

Requisitos de calificación de HSM

AWS IoT Greengrass proporciona el componente de proveedor PKCS #11 para integrarlo con el módulo de seguridad de hardware (HSM) PKCS del dispositivo. La configuración del HSM depende del dispositivo y del módulo HSM que haya elegido. Siempre que se proporcione la configuración de HSM esperada, tal como se documenta en los ajustes de configuración de IDT, IDT dispondrá de la información necesaria para realizar esta prueba de calificación de característica opcional.

Configure los ajustes de IDT para ejecutar el conjunto de AWS IoT Greengrass cualificación

Antes de ejecutar las pruebas, debe configurar los ajustes de las AWS credenciales y los dispositivos del ordenador host.

Configure AWS las credenciales en config.json

Debe configurar sus credenciales de usuario de IAM en el archivo *<device_tester_extract_location>*`/configs/config.json`. Utilice las credenciales del usuario de IDT para AWS IoT Greengrass V2 creado enthe section called “Cree y configure un Cuenta de AWS”. Puede especificar sus credenciales de una de las dos formas siguientes:

- En un archivo de credenciales
- Como variables de entorno.

Configure AWS las credenciales con un archivo de credenciales

IDT utiliza el mismo archivo de credenciales que la AWS CLI. Para obtener más información, consulte Configuración y archivos de credenciales.

La ubicación del archivo de credenciales varía en función del sistema operativo que utilice:

- macOS, Linux: `~/.aws/credentials`
- Windows: `C:\Users\`*`UserName`*`\.aws\credentials`

Añada sus AWS credenciales al `credentials` archivo en el siguiente formato:

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

Para configurar IDT para AWS IoT Greengrass V2 de modo que utilice AWS las credenciales del `credentials` archivo, edítelo `config.json` de la siguiente manera:

```
{
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

> **Note**
>
> Si no usa el `default` AWS perfil, asegúrese de cambiar el nombre del perfil en el `config.json` archivo. Para obtener más información, consulte Perfiles con nombre.

Configure AWS las credenciales con variables de entorno

Las variables de entorno son las variables que mantiene el sistema operativo y utilizan los comandos del sistema. No se guardan si cierra la sesión de SSH. IDT para AWS IoT Greengrass V2 puede usar las variables de `AWS_SECRET_ACCESS_KEY` entorno `AWS_ACCESS_KEY_ID` y para almacenar sus AWS credenciales.

Para establecer estas variables en Linux, MacOS, o Unix, utilice export:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Para establecer estas variables en Windows, utilice set:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Para configurar IDT para utilizar las variables de entorno, edite la sección `auth` de su archivo `config.json`. A continuación se muestra un ejemplo:

```
{
```

```
  "awsRegion": "region",
  "auth": {
    "method": "environment"
  }
}
```

Configurar device.json

> **Note**
>
> La versión 4.9.3 de IDT permite probar las características `ml`, `docker` y `streamManagement`. La versión 4.9.4 de IDT y las versiones posteriores admiten las pruebas de `docker`. Si no desea probar estas características, defina el valor correspondiente en no.

Además de AWS las credenciales, IDT para AWS IoT Greengrass V2 necesita información sobre los dispositivos en los que se ejecutan las pruebas. Algunos ejemplos de información serían la dirección IP, la información de inicio de sesión, el sistema operativo y la arquitectura de la CPU.

Debe proporcionar esta información utilizando la plantilla `device.json` ubicada en *<device_tester_extract_location>*`/configs/device.json`:

IDT v4.9.3

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
        "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
```

```
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`sku`

Un valor alfanumérico que identifica de forma única el dispositivo a prueba. El SKU se utiliza para realizar un seguimiento de placas cualificadas.

> **Note**
>
> Si quieres incluir tu dispositivo en el catálogo de AWS Partner dispositivos, el SKU que especifiques aquí debe coincidir con el SKU que utilices en el proceso de publicación.

`features`

Una matriz que contenga las características compatibles del dispositivo. Todas las características son obligatorias.

`arch`

Las arquitecturas de sistemas operativos compatibles que valida la ejecución de la prueba. Los valores válidos son:

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`ml`

Valida que el dispositivo cumpla con todos los requisitos técnicos necesarios para utilizar los componentes AWS de aprendizaje automático (ML) proporcionados.

Al habilitar esta función, también se valida que el dispositivo pueda realizar inferencias de aprendizaje automático mediante los marcos Deep Learning Runtime y TensorFlow Lite ML.

Los valores válidos son cualquier combinación de `dlr` y `tensorflowlite` o no.

`docker`

Valida que el dispositivo cumpla con todas las dependencias técnicas necesarias para utilizar el componente Docker Application AWS Manager () proporcionado. `aws.greengrass.DockerApplicationManager`

Al habilitar esta característica, también se valida que el dispositivo pueda descargar una imagen de contenedor de Docker desde Amazon ECR.

Los valores válidos son cualquier combinación de `yes` o no.

`streamManagement`

Valida que los dispositivos puedan descargar, instalar y ejecutar el administrador de flujos de AWS IoT Greengrass.

Los valores válidos son cualquier combinación de `yes` o no.

`hsi`

Valida que el dispositivo pueda autenticar las conexiones a los AWS IoT Greengrass servicios mediante una clave privada AWS IoT y un certificado almacenados en un módulo de seguridad de hardware (HSM). Esta prueba también verifica que el componente del proveedor PKCS #11 AWS proporcionado pueda interactuar con el HSM mediante una biblioteca PKCS #11 proporcionada por el proveedor. Para obtener más información, consulte Integración de la seguridad de hardware.

Los valores válidos son `hsm` o no.

> **Note**
>
> La prueba de `hsi` solo está disponible con la versión 4.9.3 de IDT y versiones posteriores.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`devices.operatingSystem`

El sistema operativo del dispositivo. Los valores admitidos son `Linux` y `Windows`.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Actualmente, el único valor admitido es `ssh` para dispositivos físicos.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.publicKeyPath`

Opcional. La ruta completa a la clave pública utilizada para autenticar las conexiones al dispositivo que se está probando.

Al especificar la `publicKeyPath`, IDT valida la clave pública del dispositivo cuando establece una conexión SSH con el dispositivo que se está probando. Si no se especifica este valor, IDT crea una conexión SSH, pero no valida la clave pública del dispositivo.

Le recomendamos encarecidamente que especifique la ruta a la clave pública y que utilice un método seguro para obtenerla. En el caso de los clientes SSH estándar basados en la línea de comandos, la clave pública se proporciona en el archivo `known_hosts`. Si especifica un archivo de clave pública independiente, este archivo debe usar el mismo formato que el archivo `known_hosts`, es decir, *`ip-address key-type public-key`*. Si hay varias entradas con la misma dirección IP, la entrada del tipo de clave utilizada por IDT debe estar antes que las demás entradas del archivo.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

IDT v4.9.4

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
```

```
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`sku`

Un valor alfanumérico que identifica de forma única el dispositivo a prueba. El SKU se utiliza para realizar un seguimiento de placas cualificadas.

> **Note**
>
> Si quieres incluir tu dispositivo en el catálogo de dispositivos, el AWS Partner SKU que especifiques aquí debe coincidir con el SKU que utilices en el proceso de publicación.

`features`

Una matriz que contenga las características compatibles del dispositivo. Todas las características son obligatorias.

`arch`

Las arquitecturas de sistemas operativos compatibles que valida la ejecución de la prueba. Los valores válidos son:

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`docker`

Valida que el dispositivo cumpla con todas las dependencias técnicas necesarias para utilizar el componente Docker application manager () AWS suministrado. `aws.greengrass.DockerApplicationManager`

Al habilitar esta característica, también se valida que el dispositivo pueda descargar una imagen de contenedor de Docker desde Amazon ECR.

Los valores válidos son cualquier combinación de `yes` o no.

`hsi`

Valida que el dispositivo pueda autenticar las conexiones a los AWS IoT Greengrass servicios mediante una clave privada AWS IoT y un certificado almacenados en un módulo de seguridad de hardware (HSM). Esta prueba también verifica que el componente del proveedor PKCS #11 AWS proporcionado pueda interactuar con el HSM mediante una

biblioteca PKCS #11 proporcionada por el proveedor. Para obtener más información, consulte Integración de la seguridad de hardware.

Los valores válidos son `hsm` o `no`.

> **Note**
>
> La prueba de `hsi` solo está disponible con la versión 4.9.3 de IDT y versiones posteriores.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`devices.operatingSystem`

El sistema operativo del dispositivo. Los valores admitidos son `Linux` y `Windows`.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Actualmente, el único valor admitido es `ssh` para dispositivos físicos.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.publicKeyPath`

Opcional. La ruta completa a la clave pública utilizada para autenticar las conexiones al dispositivo que se está probando.

Al especificar la `publicKeyPath`, IDT valida la clave pública del dispositivo cuando establece una conexión SSH con el dispositivo que se está probando. Si no se especifica este valor, IDT crea una conexión SSH, pero no valida la clave pública del dispositivo.

Le recomendamos encarecidamente que especifique la ruta a la clave pública y que utilice un método seguro para obtenerla. En el caso de los clientes SSH estándar basados en la línea de comandos, la clave pública se proporciona en el archivo `known_hosts`. Si especifica un archivo de clave pública independiente, este archivo debe usar el mismo formato que el archivo `known_hosts`, es decir, *`ip-address key-type public-key`*. Si hay varias entradas con la misma dirección IP, la entrada del tipo de clave utilizada por IDT debe estar antes que las demás entradas del archivo.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

Configuración de userdata.json

IDT para la AWS IoT Greengrass versión 2 también necesita información adicional sobre la ubicación del software y los artefactos de prueba. AWS IoT Greengrass

Debe proporcionar esta información utilizando la plantilla `userdata.json` ubicada en *<device_tester_extract_location>*`/configs/userdata.json`:

```
{
    "TempResourcesDirOnDevice": "/path/to/temp/folder",
    "InstallationDirRootOnDevice": "/path/to/installation/folder",
    "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
    "PreInstalled": "yes/no",
    "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
 "hsm": {
        "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
        "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
        "slotId": "slot-id",
        "slotLabel": "slot-label",
        "slotUserPin": "slot-pin",
        "keyLabel": "key-label",
        "preloadedCertificateArn": "certificate-arn"
        "rootCA": "path/to/root-ca"
    }
}
```

Todas las propiedades que contienen valores son obligatorios tal y como se describe aquí:

`TempResourcesDirOnDevice`

La ruta completa a una carpeta temporal del dispositivo que se está probando en la que se almacenan los artefactos de prueba. Asegúrese de que no se requieren permisos de sudo para escribir en este directorio.

> **Note**
>
> IDT borra el contenido de esta carpeta cuando termina de ejecutar una prueba.

`InstallationDirRootOnDevice`

La ruta completa a la carpeta del dispositivo en la que se va a instalar AWS IoT Greengrass. Para PreInstalled Greengrass, esta es la ruta al directorio de instalación de Greengrass.

Debe establecer los permisos de archivo necesarios para esta carpeta. Ejecute el siguiente comando para cada carpeta de la ruta de instalación.

```
sudo chmod 755 folder-name
```

`GreengrassNucleusZip`

La ruta completa al archivo ZIP (`greengrass-nucleus-latest.zip`) del núcleo de Greengrass en el equipo host. Este campo no es obligatorio para realizar pruebas con PreInstalled Greengrass.

> **Note**
>
> Para obtener información sobre las versiones compatibles del núcleo de Greengrass para IDT, consulte. AWS IoT GreengrassÚltima versión de IDT para V2 AWS IoT Greengrass Para descargar el software Greengrass más reciente, consulte Descargar el AWS IoT Greengrass software.

`PreInstalled`

Esta característica solo está disponible para la versión 4.5.8 de IDT y versiones posteriores en dispositivos Linux.

(Opcional) Si el valor es*yes*, IDT asumirá que la ruta mencionada es el directorio en `InstallationDirRootOnDevice` el que está instalado Greengrass.

Para obtener más información acerca de cómo instalar Greengrass en el dispositivo, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento automático de recursos. Si la instalación se realiza con aprovisionamiento manual, incluya el paso «Añadir AWS IoT el elemento a un grupo de elementos nuevo o existente» al crear un AWS IoT elemento manualmente. IDT supone que el elemento y el grupo de elementos se crean durante la configuración de la instalación. Asegúrese de que estos valores se reflejen en el archivo `effectiveConfig.yaml`. IDT busca el archivo `effectiveConfig.yaml` que aparece en `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`.

Para ejecutar pruebas con HSM, asegúrese de que el campo `aws.greengrass.crypto.Pkcs11Provider` esté actualizado en `effectiveConfig.yaml`.

`GreengrassV2TokenExchangeRole`

(Opcional) El rol de IAM personalizado que desea utilizar como rol de intercambio de token que el dispositivo objeto de prueba asume para interactuar con los recursos de AWS .

> **Note**
>
> IDT utiliza este rol de IAM personalizado en lugar de crear el rol de intercambio de token predeterminado durante la ejecución de la prueba. Si usa un rol personalizado, puede actualizar los permisos de IAM del usuario de prueba para excluir la declaración `iamResourcesUpdate` que permite al usuario crear y eliminar roles y políticas de IAM.

Para obtener más información sobre cómo crear un rol de IAM personalizado como rol de intercambio de tokens, consulte Configuración de un rol de intercambio de token personalizado.

`hsm`

Esta característica está disponible para la versión 4.5.1 de IDT y versiones posteriores.

(Opcional) La información de configuración para realizar pruebas con un módulo de seguridad de hardware (HSM) de AWS IoT Greengrass . De lo contrario, la propiedad `hsm` debe omitirse. Para obtener más información, consulte Integración de la seguridad de hardware.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

> **Warning**
>
> La configuración del HSM puede considerarse información confidencial si IDT y otro sistema comparten el módulo de seguridad de hardware. En esta situación, puede evitar proteger estos valores de configuración en texto simple almacenándolos en un AWS parámetro del almacén SecureString de parámetros y configurando IDT para que los recupere durante la ejecución de la prueba. Para obtener más información, consulte ???

`hsm.greengrassPkcsPluginJar`

La ruta completa al componente del proveedor PKCS #11 que se descarga en la máquina host de IDT. AWS IoT Greengrass proporciona este componente como un archivo JAR

que puede descargar para especificarlo como complemento de aprovisionamiento durante la instalación. Puede descargar la última versión del archivo JAR del componente en la siguiente URL: https://d2s8p88vqu9w66.cloudfront. net/releases/Pkcs11Provider/aws.greengrass.crypto.pkcs11Provider-latest.jar.

`hsm.pkcs11ProviderLibrary`

La ruta completa a la biblioteca PKCS #11 que proporciona el proveedor del módulo de seguridad de hardware (HSM) para interactuar con el HSM.

`hsm.slotId`

El identificador de ranura que se utiliza para identificar la ranura del HSM en la que se cargan la clave y el certificado.

`hsm.slotLabel`

La etiqueta de ranura que se utiliza para identificar la ranura del HSM en la que se cargan la clave y el certificado.

`hsm.slotUserPin`

El PIN de usuario que IDT utiliza para autenticar el software principal en el HSM. AWS IoT Greengrass

> **Note**
>
> La práctica recomendada de seguridad es no utilizar el mismo PIN de usuario en dispositivos de producción.

`hsm.keyLabel`

Es la etiqueta que se utiliza para identificar la clave en el módulo de hardware. Tanto la clave como el certificado deben usar la misma etiqueta de clave.

`hsm.preloadedCertificateArn`

El nombre de recurso de Amazon (ARN) del certificado del dispositivo cargado en la nube de AWS IoT .

Debe haber generado previamente este certificado con la clave del HSM, haberlo importado a su HSM y haberlo subido a la nube. AWS IoT Para obtener información sobre cómo generar e importar el certificado, consulte la documentación de su HSM.

Debe cargar el certificado en la misma cuenta y región que proporcionó en config.json. . Para obtener más información sobre cómo cargar el certificado a AWS IoT, consulte Registrar un certificado de cliente manualmente en la AWS IoT Guía para desarrolladores.

`hsm.rootCAPath`

(Opcional) La ruta completa de la máquina host de IDT a la entidad de certificación (CA) raíz que firmó el certificado. Esto es obligatorio si el certificado creado en el HSM no está firmado por la CA raíz de Amazon.

Obtenga la configuración del almacén de parámetros AWS

AWS IoT El Device Tester (IDT) incluye una función opcional para obtener los valores de configuración del almacén de parámetros de AWS Systems Manager. AWS El almacén de parámetros permite almacenar las configuraciones de forma segura y cifrada. Cuando está configurado, IDT puede recuperar los parámetros del almacén de AWS parámetros en lugar de almacenarlos en texto plano dentro del archivo. `userdata.json` Esto resulta útil para cualquier dato confidencial que deba almacenarse cifrado, como contraseñas, números PIN y otros datos secretos.

1. Para utilizar esta función, debe actualizar los permisos utilizados al crear su usuario de IDT para permitir la GetParameter acción en los parámetros para los que IDT está configurado. El siguiente es un ejemplo de una declaración de permiso que se puede agregar al usuario de IDT. Para obtener más información, consulte la Guía del usuario de AWS Systems Manager.

```
{
       "Sid":"parameterStoreResources",
       "Effect": "Allow",
        "Action": [
            "ssm:GetParameter"
        ],
        "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
}
```

El permiso anterior está configurado para permitir la obtención de todos los parámetros cuyo nombre comience por `IDT`, mediante el carácter comodín `*`. Debe personalizarlo según sus necesidades para que IDT tenga acceso a cualquier parámetro configurado en función del nombre de los parámetros que esté utilizando.

2. Debe almacenar los valores de configuración en AWS Paramater Store. Esto se puede hacer desde la AWS consola o desde la AWS CLI. AWS Parameter Store le permite elegir un almacenamiento cifrado o no cifrado. Para almacenar valores confidenciales como secretos, contraseñas y pines, debe usar la opción cifrada, que es un tipo de parámetro de SecureString. Para cargar un parámetro mediante la AWS CLI, puede utilizar el siguiente comando:

```
aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
 SecureString
```

Puede confirmar que un parámetro está almacenado mediante el siguiente comando. (Opcional) Utilice la `--with-decryption` marca para obtener un parámetro descifrado. SecureString

```
aws ssm get-parameter --name IDT-example-name
```

El uso de la AWS CLI cargará el parámetro en la AWS región del usuario de CLI actual e IDT obtendrá los parámetros de la región configurada. `config.json` Para comprobar su región desde la CLI de AWS , utilice lo siguiente:

```
aws configure get region
```

3. Una vez que tenga un valor de configuración en la AWS nube, podrá actualizar cualquier valor de la configuración de IDT para obtenerlo de la nube. AWS Para ello, utilice un marcador de posición en la configuración de IDT del formulario `{{AWS.Parameter.parameter_name}}` para buscar el parámetro con ese nombre en el almacén de parámetros. AWS

Por ejemplo, supongamos que quiere usar el parámetro `IDT-example-name` del paso 2 como HSM keyLabel en su configuración de HSM. Para ello, puede actualizar su `userdata.json` de la siguiente manera:

```
"hsm": {
        "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
        [...]
    }
```

IDT obtendrá el valor de este parámetro en el tiempo de ejecución que se configuró como `IDT-example-value` en el paso 2. Esta configuración es similar a la configuración, `"keyLabel": "IDT-example-value"` pero, en cambio, ese valor se almacena cifrado en la nube. AWS

Ejecute el conjunto AWS IoT Greengrass de calificaciones

Después de configurar los ajustes necesarios, puede iniciar las pruebas. El tiempo de ejecución del conjunto completo de pruebas depende de su hardware. Como referencia, se tarda aproximadamente 30 minutos en completar el conjunto de pruebas completo en una unidad Raspberry Pi 3B.

Utilice el siguiente comando `run-suite` para ejecutar un conjunto de pruebas.

```
devicetester_[linux | mac | win]_x86-64 run-suite  \\
    --suite-id suite-id  \\
    --group-id group-id  \\
    --pool-id your-device-pool \\
    --test-id test-id  \\
    --update-idt y|n  \\
    --userdata userdata.json
```

Todas las opciones son opcionales. Por ejemplo, puede omitir `pool-id` solo si tiene un grupo de dispositivos, que es un conjunto de dispositivos idénticos, definido en el archivo `device.json`. O bien, puede omitir `suite-id` si desea ejecutar la última versión del conjunto de pruebas en la carpeta `tests`.

> **Note**
>
> IDT le pregunta si está disponible en línea una versión más reciente del conjunto de pruebas. Para obtener más información, consulte the section called “Versiones del conjunto de pruebas”.

Ejemplos de comandos para ejecutar el conjunto de calificaciones

Los comandos de ejemplo siguientes muestran cómo ejecutar las pruebas de calificación para un grupo de dispositivos. Para obtener más información acerca de `run-suite` y otros comandos IDT, consulte the section called “Comandos de IDT”.

Utilice el siguiente comando para ejecutar todos los grupos de prueba en un conjunto especificado. Utilice el comando `list-suites` para enumerar los conjuntos de pruebas que se encuentran en la carpeta `tests`.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
```

```
    --suite-id GGV2Q_1.0.0 \
    --pool-id <pool-id> \
    --userdata userdata.json
```

Utilice el siguiente comando para ejecutar un grupo de prueba específico en un conjunto de pruebas. Utilice el comando `list-groups` para enumerar los grupos de prueba en un conjunto de pruebas.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --group-id <group-id> \
    --pool-id <pool-id> \
    --userdata userdata.json
```

Utilice el siguiente comando para ejecutar un caso de prueba específico en un grupo de pruebas.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id> \
    --userdata userdata.json
```

Utilice el siguiente comando para ejecutar varios casos de prueba en un grupo de pruebas.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id1>,<test-id2>
    --userdata userdata.json
```

Utilice el comando para enumerar los casos de prueba en un grupo de pruebas.

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

Le recomendamos que ejecute el conjunto completo de pruebas de calificación, que ejecuta las dependencias de los grupos de pruebas en el orden correcto. Si decide ejecutar grupos de pruebas específicos, le recomendamos que primero ejecute el grupo de pruebas del verificador de dependencias para asegurarse de que todas las dependencias de Greengrass estén instaladas antes de ejecutar los grupos de pruebas relacionados. Por ejemplo:

- Ejecute `coredependencies` antes de ejecutar los grupos de prueba de cualificación del núcleo.

Comandos IDT para AWS IoT Greengrass V2

Los comandos IDT se encuentran en el directorio *<device-tester-extract-location>*/bin. Para ejecutar un conjunto de pruebas, debe proporcionar el comando en el siguiente formato:

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto de prueba determinado.

`list-suites`

Muestra los conjuntos de prueba disponibles.

`list-supported-products`

Muestra los productos compatibles, en este caso AWS IoT Greengrass las versiones, y las versiones del conjunto de pruebas de la versión actual de IDT.

`list-test-cases`

Enumera los casos de prueba en un grupo de prueba determinado. Se admite la siguiente opción:

- `group-id`. El grupo de pruebas que se va a buscar. Esta opción es necesaria y debe especificar un solo grupo.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos. Las siguientes son algunas de las opciones admitidas:

- `suite-id`. La versión del conjunto de pruebas que se va a ejecutar. Si no se especifica, IDT utiliza la versión más reciente de la carpeta `tests`.
- `group-id`. Los grupos de pruebas que se van a ejecutar, como una lista separada por comas. Si no se especifica, IDT ejecuta todos los grupos de pruebas adecuados del conjunto de pruebas en función de los ajustes configurados en `device.json`. IDT no ejecuta ningún grupo de pruebas que el dispositivo no admita en función de los ajustes configurados, incluso si esos grupos de prueba están especificados en la lista `group-id`.
- `test-id`. Los casos de prueba que se van a ejecutar, como una lista separada por comas. Cuando se especifique, `group-id` debe especificar un solo grupo.
- `pool-id`. El grupo de dispositivos que se va a probar. Debe especificar un grupo si tiene varios grupos de dispositivos definidos en el archivo `device.json`.

- `stop-on-first-failure`. Configura IDT para detener la ejecución en el primer error. Esta opción debe utilizarse con `group-id` para depurar los grupos de prueba especificados. No utilice esta opción cuando ejecute un conjunto de pruebas completo para generar un informe de cualificación.
- `update-idt`. Establece la respuesta para la petición de actualización de IDT. La respuesta `Y` detiene la ejecución de la prueba si IDT detecta que hay una versión más reciente. La respuesta `N` continúa con la ejecución de la prueba.
- `userdata`. La ruta completa al archivo `userdata.json` que contiene información sobre las rutas de los artefactos de prueba. Esta opción es necesaria para el comando `run-suite`. El `userdata.json` archivo debe estar ubicado en el directorio *`devicetester_extract_location`* /devicetester_ggv2_ /configs/*`[win|mac|linux]`*.

Para obtener más información acerca de `run-suite` las opciones, utilice la opción `help`:

```
devicetester_[linux | mac | win]_x86-64 run-suite -h
```

Descripción de los resultados y de los registros

En esta sección se describe cómo ver e interpretar registros e informes de resultados de IDT.

Para solucionar errores, consulte Solución de problemas de IDT para V2 AWS IoT Greengrass.

Ver los resultados

Mientras ejecuta, IDT escribe errores en la consola, en archivos de registro y en informes de prueba. Una vez que IDT completa el conjunto de pruebas de cualificación, genera dos informes de prueba. Estos archivos de informes están ubicados en *`<device-tester-extract-location>`*`/results/`*`<execution-id>`*`/`. Ambos informes capturan los resultados de la ejecución del conjunto de pruebas de cualificación.

El `awsiotdevicetester_report.xml` es el informe de prueba de cualificación que envía a AWS para mostrar su dispositivo en el AWS Partner de Device Catalog. El informe contiene los componentes siguientes:

- La versión de IDT.
- La versión AWS IoT Greengrass que se ha probado.
- El SKU y el grupo de dispositivos especificado en el archivo `device.json`.

- Las características del grupo de dispositivos especificado en el archivo `device.json`.
- El resumen de agregación de los resultados de las pruebas.
- Un desglose de los resultados de las pruebas por bibliotecas que se probaron en función de las características del dispositivo, como acceso a recursos locales, shadow y MQTT.

El informe `GGV2Q_Result.xml` está en formato XML JUnit. Puede integrarlo en plataformas de integración/implementación continua como Jenkins, Bamboo, etc. El informe contiene los componentes siguientes:

- Resumen de agregación de los resultados de pruebas.
- Desglose de resultados de pruebas por funcionalidad de AWS IoT Greengrass probada.

Interpretación de los resultados AWS IoT Device Tester

La sección de informe en `awsiotdevicetester_report.xml` o `awsiotdevicetester_report.xml` enumera las pruebas que se ejecutaron y los resultados.

La primera etiqueta XML `<testsuites>` contiene el resumen de la ejecución de las pruebas. Por ejemplo:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atributos que se utilizan en la etiqueta **`<testsuites>`**

`name`

El nombre del grupo de prueba.

`time`

El tiempo, en segundos, que se ha tardado en ejecutar el conjunto de calificación.

`tests`

El número de pruebas que se realizaron.

`failures`

El número de pruebas que se ejecutaron, pero que no se superaron.

`errors`

El número de pruebas que IDT no ha podido ejecutar.

`disabled`

Ignore este atributo. No se utiliza.

El archivo `awsiotdevicetester_report.xml` contiene una etiqueta `<awsproduct>` que tiene información sobre el producto que se está probando y las características del producto que se han validado después de ejecutar un conjunto de pruebas.

Atributos que se utilizan en la etiqueta **`<awsproduct>`**

`name`

El nombre del producto que se está probando.

`version`

La versión del producto que se está probando.

`features`

Las características validadas. Las características marcadas como `required` son necesarias para solicitar la cualificación de la placa. En el siguiente fragmento se muestra cómo aparece esta información en el archivo `awsiotdevicetester_report.xml`.

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></feature>
```

Si no hay errores de pruebas para las características requeridas, el dispositivo cumple los requisitos técnicos para ejecutar AWS IoT Greengrass y puede interoperar con servicios de AWS IoT. Si quiere mostrar su dispositivo en el AWS Partner Device Catalog, puede utilizar este informe como prueba de cualificación.

Si se producen errores en pruebas, puede identificar la prueba fallido revisando las etiquetas XML `<testsuites>`. Las etiquetas XML `<testsuite>` dentro de la etiqueta `<testsuites>` muestran el resumen del resultado de la prueba de un grupo de prueba. Por ejemplo:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

El formato es similar a la etiqueta `<testsuites>`, pero con un atributo `skipped` que no se utiliza y que se puede pasar por alto. Dentro de cada etiqueta XML `<testsuite>`, hay etiquetas `<testcase>` para cada prueba ejecutada para un grupo de prueba. Por ejemplo:

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security
 Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled
 and following changes are made:Add CIS conn info and Add another CIS conn info"
 attempts="1"></testcase>>
```

Atributos que se utilizan en la etiqueta **`<testcase>`**

`name`

El nombre de la prueba.

`attempts`

El número de veces que IDT ha ejecutado el caso de prueba.

Cuando una prueba genera un error o si se produce un error, las etiquetas `<failure>` o `<error>` se agregan a la etiqueta `<testcase>` con información para la resolución de problemas. Por ejemplo:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Visualización de registros

IDT genera registros a partir de la ejecución de pruebas en *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs`. Se generan dos conjuntos de registros:

`test_manager.log`

Registros generados desde el componente Administrador de pruebas de AWS IoT Device Tester (por ejemplo, registros relacionados con la configuración, la secuenciación de pruebas y la generación de informes).

`<test-case-id>``.log (for example, lambdaDeploymentTest.log)`

Los registros del grupo de prueba, incluidos los registros del dispositivo a prueba. A partir de IDT versión 4.2.0, IDT agrupa los registros de pruebas de cada caso de prueba en una carpeta

<test-case-id> separada dentro del directorio *<devicetester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>/*.

Uso de IDT para desarrollar y ejecutar sus propios conjuntos de pruebas

A partir de la versión 4.0.1 de IDT, IDT para AWS IoT Greengrass V2 combina una configuración y un formato de resultados estandarizados con un entorno de conjuntos de pruebas que le permite desarrollar conjuntos de pruebas personalizados para sus dispositivos y el software de los dispositivos. Puede añadir pruebas personalizadas para su propia validación interna o proporcionárselas a sus clientes para la verificación de los dispositivos.

Utilice IDT para desarrollar y ejecutar conjuntos de pruebas personalizados, de la siguiente manera:

Para desarrollar conjuntos de pruebas personalizados

- Cree conjuntos de pruebas con lógica de prueba personalizada para el dispositivo Greengrass que desee probar.
- Proporcione a IDT sus conjuntos de pruebas personalizados para los corredores de pruebas. Incluya información sobre las configuraciones de configuración específicas de sus conjuntos de pruebas.

Ejecución de conjuntos de pruebas personalizados

- Configure el dispositivo que desea probar.
- Implemente las configuraciones requeridas por los conjuntos de pruebas que desee utilizar.
- Utilice IDT para ejecutar sus conjuntos de pruebas personalizados.
- Vea los resultados de las pruebas y los registros de ejecución de las pruebas realizadas por IDT.

Descargue la última versión de for AWS IoT Device Tester AWS IoT Greengrass

Descargue la última versión de IDT y extraiga el software en una ubicación (*<device-tester-extract-location>*) del sistema de archivos en la que tenga permisos de lectura/escritura.

> **Note**
>
> IDT no admite la ejecución por parte de varios usuarios desde una ubicación compartida, como un directorio NFS o una carpeta compartida de red de Windows. Le recomendamos que extraiga el paquete IDT en una unidad local y ejecute el binario IDT en su estación de trabajo local.
> Windows tiene una limitación de longitud de ruta de 260 caracteres. Si utiliza Windows, extraiga IDT en un directorio raíz como `C:\` o `D:\` para mantener las rutas por debajo del límite de 260 caracteres.

Flujo de trabajo de creación de conjuntos de prueba

Los conjuntos de pruebas se componen de tres tipos de archivos:

- Archivos de configuración que proporcionan a IDT información sobre cómo ejecutar el conjunto de pruebas.
- Archivos ejecutables de prueba que IDT utiliza para ejecutar los casos de prueba.
- Archivos adicionales necesarios para ejecutar las pruebas.

Complete los siguientes pasos básicos para crear pruebas de IDT personalizadas:

1. Cree archivos de configuración para su conjunto de pruebas.
2. Cree ejecutables de casos de prueba que contengan la lógica de prueba de su conjunto de pruebas.
3. Verifique y documente la información de configuración necesaria para que los ejecutores de pruebas ejecuten el conjunto de pruebas.
4. Compruebe que IDT pueda ejecutar su conjunto de pruebas y producir los resultados de las pruebas según lo esperado.

Para crear rápidamente un conjunto personalizado de muestra y ejecutarlo, siga las instrucciones que se indican en Tutorial: crear y ejecutar el ejemplo del conjunto de pruebas de IDT.

Para empezar a crear un conjunto de pruebas personalizado en Python, consulte Tutorial: Desarrollar un conjunto de pruebas de IDT sencillo.

Tutorial: crear y ejecutar el ejemplo del conjunto de pruebas de IDT

La AWS IoT Device Tester descarga incluye el código fuente de un conjunto de pruebas de muestra. Puede completar este tutorial para crear y ejecutar el conjunto de pruebas de ejemplo para comprender cómo puede usar IDT AWS IoT Greengrass para ejecutar conjuntos de pruebas personalizados.

En este tutorial, debe completar los siguientes pasos:

1. Creación del conjunto de pruebas de muestra
2. Uso de IDT para ejecutar el conjunto de pruebas de muestra

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Requisitos del equipo host
  - Versión más reciente de AWS IoT Device Tester
  - Python 3.7 o posterior

    Para comprobar la versión de Python instalada en el equipo, ejecute el siguiente comando:

    ```
    python3 --version
    ```

    En Windows, si el uso de este comando devuelve un error, utilice `python --version` en su lugar. Si el número de versión devuelto es 3.7 o superior, ejecute el siguiente comando en una terminal de Powershell para configurar `python3` como alias para el comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Si no se devuelve información sobre la versión o si la versión es inferior a 3.7, siga las instrucciones que se indican en Descarga de Python para instalar Python 3.7 o superior. Para obtener más información, consulte la documentación de Python.
  - urllib3

    Para comprobar que `urllib3` se ha instalado correctamente, ejecute el siguiente comando:

```
python3 -c 'import urllib3'
```

Si `urllib3` no está instalado, ejecute el siguiente comando para instalarlo:

```
python3 -m pip install urllib3
```

- Requisitos de los dispositivos
  - Un dispositivo con un sistema operativo Linux y una conexión de red a la misma red que el equipo host.

    Le recomendamos que utilice un Raspberry Pi con el sistema operativo Raspberry Pi. Asegúrese de que tiene configurado SSH en el Raspberry Pi para conectarse de forma remota.

Configuración de la información del dispositivo para IDT

Configure la información del dispositivo para que IDT ejecute la prueba. Debe actualizar la plantilla `device.json` ubicada en la carpeta *<device-tester-extract-location>*`/configs` con la siguiente información.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
```

```
      }
    ]
  }
]
```

En el objeto `devices`, indique la siguiente información:

`id`

Un identificador único definido por el usuario para el dispositivo.

`connectivity.ip`

La dirección IP del dispositivo.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH al dispositivo.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.user`

El nombre de usuario utilizado para iniciar sesión en el dispositivo.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`devices.connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Creación del conjunto de pruebas de muestra

La carpeta *<device-tester-extract-location>*`/samples/python` contiene ejemplos de archivos de configuración, código fuente y el SDK de cliente de IDT que puede combinar en un conjunto de pruebas mediante los scripts de creación proporcionados. El siguiente árbol de directorios muestra la ubicación de estos archivos de ejemplo:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Para crear el conjunto de pruebas, ejecute los siguientes comandos en el equipo host:

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
```

```
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

Esto crea el conjunto de pruebas de muestra en la carpeta `IDTSampleSuitePython_1.0.0`, dentro de la carpeta *<device-tester-extract-location>*`/tests`. Revise los archivos de la carpeta `IDTSampleSuitePython_1.0.0` para ver cómo está estructurado el conjunto de pruebas de muestra y consulte varios ejemplos de ejecutables de casos de prueba y archivos JSON de configuración de pruebas.

> **Note**
>
> El conjunto de pruebas de muestra incluye el código fuente de Python. No incluya información confidencial en el código del conjunto de pruebas.

Siguiente paso: Uso de IDT para ejecutar el conjunto de pruebas de muestra que creó.

Uso de IDT para ejecutar el conjunto de pruebas de muestra

Para ejecutar el conjunto de pruebas de muestra, ejecute los siguientes comandos en el equipo host:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT ejecuta el conjunto de pruebas de muestra y transmite los resultados a la consola. Cuando la prueba termine de ejecutarse, verá la siguiente información:

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    sample_group:       PASSED
```

```
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Resolución de problemas

Utilice la siguiente información para resolver cualquier problema que pueda surgir al completar el tutorial.

El caso de prueba no se ejecuta correctamente

Si la prueba no se ejecuta correctamente, IDT transmite los registros de errores a la consola para ayudarle a solucionar los problemas de la prueba. Asegúrese de que cumple con todos los requisitos previos de este tutorial.

No se puede conectar al dispositivo que se está probando

Compruebe lo siguiente:

- El archivo `device.json` contiene la dirección IP, el puerto y la información de autenticación correctos.
- Puede conectarse a su dispositivo a través de SSH desde su equipo host.

Tutorial: Desarrollar un conjunto de pruebas de IDT sencillo

Un conjunto de pruebas combina lo siguiente:

- Ejecutables de prueba que contienen la lógica de prueba
- Archivos de configuración que describen el conjunto de pruebas

Este tutorial le muestra cómo usar IDT AWS IoT Greengrass para desarrollar un conjunto de pruebas de Python que contenga un único caso de prueba. En este tutorial, debe completar los siguientes pasos:

1. Creación de un directorio del conjunto de pruebas
2. Creación de archivos de configuración

3. Creación del ejecutable del caso de prueba
4. Ejecución del conjunto de pruebas

Requisitos previos

Necesitará lo siguiente para completar este tutorial:

- Requisitos del equipo host
  - Última versión de AWS IoT Device Tester
  - Python 3.7 o posterior

    Para comprobar la versión de Python instalada en el equipo, ejecute el siguiente comando:

    ```
    python3 --version
    ```

    En Windows, si el uso de este comando devuelve un error, utilice `python --version` en su lugar. Si el número de versión devuelto es 3.7 o superior, ejecute el siguiente comando en una terminal de Powershell para configurar `python3` como alias para el comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Si no se devuelve información sobre la versión o si la versión es inferior a 3.7, siga las instrucciones que se indican en Descarga de Python para instalar Python 3.7 o superior. Para obtener más información, consulte la documentación de Python.
  - urllib3

    Para comprobar que `urllib3` se ha instalado correctamente, ejecute el siguiente comando:

    ```
    python3 -c 'import urllib3'
    ```

    Si `urllib3` no está instalado, ejecute el siguiente comando para instalarlo:

    ```
    python3 -m pip install urllib3
    ```
- Requisitos de los dispositivos
  - Un dispositivo con un sistema operativo Linux y una conexión de red a la misma red que el equipo host.

Le recomendamos que utilice un Raspberry Pi con el sistema operativo Raspberry Pi. Asegúrese de que tiene configurado SSH en el Raspberry Pi para conectarse de forma remota.

Creación de un directorio del conjunto de pruebas

IDT separa de forma lógica los casos de prueba en grupos de pruebas dentro de cada conjunto de pruebas. Cada caso de prueba debe estar dentro de un grupo de prueba. Para este tutorial, cree una carpeta llamada `MyTestSuite_1.0.0` y cree el siguiente árbol de directorios dentro de esta carpeta:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Creación de archivos de configuración

El conjunto de pruebas debe contener los siguientes archivos de configuración necesarios:

Archivos de configuración necesarios

`suite.json`

Contiene información sobre el conjunto de pruebas. Consulte Configuración de suite.json.

`group.json`

Contiene información sobre un grupo de pruebas. Debe crear un archivo `group.json` para cada grupo de pruebas de su conjunto de pruebas. Consulte Configuración de group.json.

`test.json`

Contiene información sobre un caso de prueba. Debe crear un archivo `test.json` para cada caso de prueba de su conjunto de pruebas. Consulte Configuración de test.json.

1. En la carpeta `MyTestSuite_1.0.0/suite`, cree un archivo `suite.json` con la estructura siguiente:

   ```
   {
       "id": "MyTestSuite_1.0.0",
   ```

```
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. En la carpeta `MyTestSuite_1.0.0/myTestGroup`, cree un archivo `group.json` con la estructura siguiente:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. En la carpeta `MyTestSuite_1.0.0/myTestGroup/myTestCase`, cree un archivo `test.json` con la estructura siguiente:

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
```

```
}
```

El árbol de directorios de la carpeta `MyTestSuite_1.0.0` debe ser similar al siguiente:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obtención del SDK de cliente de IDT

Utilice el SDK de cliente de IDT para permitir que IDT interactúe con el dispositivo que se está probando e informe de los resultados de las pruebas. Para este tutorial, utilizará la versión Python del SDK.

Desde la carpeta *<device-tester-extract-location>*`/sdks/python/`, copie la carpeta `idt_client` a su carpeta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`.

Para comprobar que el SDK se ha copiado correctamente, ejecute el siguiente comando.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Creación del ejecutable del caso de prueba

Los ejecutables del caso de prueba contienen la lógica de prueba que desea ejecutar. Un conjunto de pruebas puede contener varios ejecutables de casos de prueba. Para este tutorial, creará solo un ejecutable de caso de prueba.

1. Cree el archivo del conjunto de pruebas.

   En la carpeta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`, cree un archivo `myTestCase.py` con el siguiente contenido:

   ```
   from idt_client import *

   def main():
   ```

```
    # Use the client SDK to communicate with IDT
    client = Client()

if __name__ == "__main__":
    main()
```

2. Utilice las funciones del SDK del cliente para añadir la siguiente lógica de prueba al archivo `myTestCase.py`:

   a. Ejecute un comando SSH en el dispositivo que se está probando.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

   b. Envíe el resultado de la prueba a IDT.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
```

```
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

Configuración de la información del dispositivo para IDT

Configure la información del dispositivo para que IDT ejecute la prueba. Debe actualizar la plantilla `device.json` ubicada en la carpeta *<device-tester-extract-location>*`/configs` con la siguiente información.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
```

```
  }
]
```

En el objeto `devices`, indique la siguiente información:

`id`

Un identificador único definido por el usuario para el dispositivo.

`connectivity.ip`

La dirección IP del dispositivo.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH al dispositivo.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.user`

El nombre de usuario utilizado para iniciar sesión en el dispositivo.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`devices.connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el dispositivo.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

> **Note**
>
> Especifique `privKeyPath` solo si `method` está establecido en `pki`
> Especifique `password` solo si `method` está establecido en `password`

Ejecución del conjunto de pruebas

Después de crear el conjunto de pruebas, querrá asegurarse de que funciona como se espera. Complete los siguientes pasos para ejecutar el conjunto de pruebas con su grupo de dispositivos existente para ello.

1. Copie su carpeta `MyTestSuite_1.0.0` en *<device-tester-extract-location>*`/tests`.
2. Ejecute los siguientes comandos :

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT ejecuta el conjunto de pruebas y transmite los resultados a la consola. Cuando la prueba termine de ejecutarse, verá la siguiente información:

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

========== Test Summary ==========
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
```

```
Tests Skipped:          0
---------------------------------
Test Groups:
    myTestGroup:        PASSED
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Resolución de problemas

Utilice la siguiente información para resolver cualquier problema que pueda surgir al completar el tutorial.

El caso de prueba no se ejecuta correctamente

Si la prueba no se ejecuta correctamente, IDT transmite los registros de errores a la consola para ayudarle a solucionar los problemas de la prueba. Antes de comprobar los registros de errores, verifique lo siguiente:

- El SDK de cliente de IDT se encuentra en la carpeta correcta, tal y como se describe en este paso.
- Cumpla con todos los requisitos previos de este tutorial.

No se puede conectar al dispositivo que se está probando

Compruebe lo siguiente:

- El archivo `device.json` contiene la dirección IP, el puerto y la información de autenticación correctos.
- Puede conectarse a su dispositivo a través de SSH desde su equipo host.

Creación de archivos de configuración para el conjunto de pruebas de IDT

En esta sección se describen los formatos en los que se crean los archivos de configuración que se incluyen al escribir un conjunto de pruebas personalizado.

Archivos de configuración necesarios

`suite.json`

Contiene información sobre el conjunto de pruebas. Consulte Configuración de suite.json.

`group.json`

Contiene información sobre un grupo de pruebas. Debe crear un archivo `group.json` para cada grupo de pruebas de su conjunto de pruebas. Consulte Configuración de group.json.

`test.json`

Contiene información sobre un caso de prueba. Debe crear un archivo `test.json` para cada caso de prueba de su conjunto de pruebas. Consulte Configuración de test.json.

Archivos de configuración opcionales

`test_orchestrator.yaml` or `state_machine.json`

Define cómo se ejecutan las pruebas cuando IDT ejecuta el conjunto de pruebas. Consulte Configuración de test_orchestrator.yaml.

> **Note**
>
> A partir de la versión 4.5.1 de IDT, se utiliza el archivo `test_orchestrator.yaml` para definir el flujo de trabajo de las pruebas. En las versiones anteriores de IDT, se utiliza el archivo `state_machine.json`. Para obtener más información sobre la máquina de estados, consulte Configure la máquina de estados IDT.

`userdata_schema.json`

Define el esquema del archivo `userdata.json` que los ejecutores de pruebas pueden incluir en su configuración de ajustes. El archivo `userdata.json` se utiliza para cualquier información de configuración adicional necesaria para ejecutar la prueba, pero que no esté presente en el archivo `device.json`. Consulte Configuración de userdata_schema.json.

Los archivos de configuración se colocan en su *<custom-test-suite-folder>*, tal y como se muestra aquí.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configuración de suite.json

El archivo `suite.json` establece las variables de entorno y determina si los datos del usuario son necesarios para ejecutar el conjunto de pruebas. Utilice la siguiente plantilla para configurar el archivo *<custom-test-suite-folder>*`/suite/suite.json`:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID único definido por el usuario para el conjunto de pruebas. El valor de `id` debe coincidir con el nombre de la carpeta del conjunto de pruebas en la que se encuentra el archivo `suite.json`. El nombre y la versión del conjunto también deben cumplir los siguientes requisitos:

- *<suite-name>* no puede contener guiones bajos.
- *<suite-version>* se indica como *x.x.x*, donde x es un número.

El ID se muestra en los informes de prueba generados por IDT.

`title`

Un nombre definido por el usuario para el producto o la característica que se está probando en este conjunto de pruebas. El nombre se muestra en la CLI de IDT para los ejecutores de pruebas.

`details`

Una descripción corta de la finalidad del conjunto de pruebas.

`userDataRequired`

Define si los ejecutores de pruebas deben incluir información personalizada en un archivo `userdata.json`. Si establece este valor en `true`, también debe incluir el archivo `userdata_schema.json` en la carpeta del conjunto de pruebas.

`environmentVariables`

Opcional. Una matriz de variables de entorno que se va a configurar para este conjunto de pruebas.

`environmentVariables.key`

El nombre de la variable de entorno.

`environmentVariables.value`

El valor de la variable de entorno.

Configuración de group.json

El archivo `group.json` define si el grupo de prueba es obligatorio u opcional. Utilice la siguiente plantilla para configurar el archivo *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json`:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
```

```
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID único definido por el usuario para el conjunto de pruebas. El valor de `id` debe coincidir con el nombre de la carpeta del grupo de pruebas en la que se encuentra el archivo `group.json` y no debe contener guiones bajos (_). El ID se utiliza en los informes de prueba generados por IDT.

`title`

Un nombre descriptivo para el grupo de prueba. El nombre se muestra en la CLI de IDT para los ejecutores de pruebas.

`details`

Una descripción corta de la finalidad del grupo de pruebas.

`optional`

Opcional. Establézcalo en `true` para mostrar este grupo de pruebas como un grupo opcional una vez que IDT termine de ejecutar las pruebas requeridas. El valor predeterminado es `false`.

Configuración de test.json

El archivo `test.json` determina los ejecutables del caso de prueba y las variables de entorno que utiliza un caso de prueba. Para obtener más información sobre cómo crear ejecutables de casos de prueba, consulte Cree ejecutables de casos de prueba de IDT.

Utilice la siguiente plantilla para configurar el archivo *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json:

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
```

```
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
                }
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID único definido por el usuario para el caso de prueba. El valor de `id` debe coincidir con el nombre de la carpeta del grupo de pruebas en la que se encuentra el archivo `test.json` y no debe contener guiones bajos (_). El ID se utiliza en los informes de prueba generados por IDT.

`title`

Un nombre descriptivo para el caso de prueba. El nombre se muestra en la CLI de IDT para los ejecutores de pruebas.

`details`

Una breve descripción de la finalidad del caso de prueba.

`requireDUT`

Opcional. Establézcalo en `true` si se requiere un dispositivo para ejecutar esta prueba; de lo contrario, establézcalo en `false`. El valor predeterminado es `true`. Los ejecutores de las pruebas configurarán los dispositivos que utilizarán para ejecutar la prueba en su archivo `device.json`.

`requiredResources`

Opcional. Una matriz que proporciona información sobre los dispositivos de recursos necesarios para ejecutar esta prueba.

`requiredResources.name`

El nombre exclusivo que se asignará al dispositivo de recursos cuando se ejecute esta prueba.

`requiredResources.features`

Una matriz de características del dispositivo de recursos definidas por el usuario.

`requiredResources.features.name`

El nombre de la característica. La característica del dispositivo para la que desea utilizar este dispositivo. Este nombre se coteja con el nombre de la característica que proporciona el ejecutor de las pruebas en el archivo `resource.json`.

`requiredResources.features.version`

Opcional. La versión de la característica. Este valor se coteja con la versión de la característica proporcionada por el ejecutor de las pruebas en el archivo `resource.json`. Si no se proporciona una versión, la característica no se comprueba. Si no se necesita un número de versión para la característica, deje este campo en blanco.

`requiredResources.features.jobSlots`

Opcional. El número de pruebas simultáneas que puede admitir esta característica. El valor predeterminado es 1. Si desea que IDT utilice distintos dispositivos para características individuales, le recomendamos que establezca este valor en 1.

`execution.timeout`

La cantidad de tiempo (en milisegundos) que IDT espera a que la prueba termine de ejecutarse. Para obtener más información sobre cómo establecer este valor, consulte Cree ejecutables de casos de prueba de IDT.

`execution.os`

Los ejecutables del caso de prueba que se ejecutarán en función del sistema operativo del equipo host que ejecuta IDT. Los valores admitidos son `linux`, `mac` y `win`.

`execution.os.cmd`

La ruta al ejecutable del caso de prueba que desea ejecutar para el sistema operativo especificado. Esta ubicación debe estar en la ruta del sistema.

`execution.os.args`

Opcional. Los argumentos que se deben proporcionar para ejecutar el ejecutable del caso de prueba.

`environmentVariables`

Opcional. Una matriz de variables de entorno definidas para este caso de prueba.

`environmentVariables.key`

El nombre de la variable de entorno.

`environmentVariables.value`

El valor de la variable de entorno.

> **Note**
>
> Si especifica la misma variable de entorno en el archivo `test.json` y en el archivo `suite.json`, el valor del archivo `test.json` tiene prioridad.

Configuración de test_orchestrator.yaml

Un orquestador de pruebas es un constructo que controla el flujo de ejecución del conjunto de pruebas. Determina el estado inicial de un conjunto de pruebas, administra las transiciones de estado en función de las reglas definidas por el usuario y continúa pasando por esos estados hasta alcanzar el estado final.

Si su conjunto de pruebas no incluye un orquestador de pruebas definido por el usuario, IDT lo generará.

El orquestador de pruebas predeterminado realiza las siguientes funciones:

- Ofrece a los ejecutores de pruebas la posibilidad de seleccionar y ejecutar grupos de pruebas específicos, en lugar de todo el conjunto de pruebas.
- Si no se seleccionan grupos de pruebas específicos, ejecuta todos los grupos de pruebas del conjunto de pruebas con asignación al azar.
- Genera informes e imprime un resumen de la consola que muestra los resultados de las pruebas de cada grupo y caso de prueba.

Para obtener más información sobre cómo funciona el orquestador de pruebas, consulte Configuración del orquestador de pruebas de IDT.

Configuración de userdata_schema.json

El archivo `userdata_schema.json` determina el esquema en el que los ejecutores de las pruebas proporcionan los datos de usuario. Los datos de usuario son necesarios si su conjunto de pruebas requiere información que no está presente en el archivo `device.json`. Por ejemplo, es posible que las pruebas necesiten credenciales de red Wi-Fi, puertos abiertos específicos o certificados que deba proporcionar un usuario. Esta información se puede proporcionar a IDT como un parámetro de entrada denominado`userdata`, cuyo valor es un archivo `userdata.json`, que los usuarios crean en su carpeta *`<device-tester-extract-location>`*`/config`. El formato del archivo `userdata.json` se basa en el archivo `userdata_schema.json` que se incluye en el conjunto de pruebas.

Para indicar que los ejecutores de pruebas deben proporcionar un archivo `userdata.json`:

1. En el archivo `suite.json`, establezca `userDataRequired` en `true`.
2. En su *`<custom-test-suite-folder>`*, cree un archivo `userdata_schema.json`.

3. Edite el archivo `userdata_schema.json` para crear un borrador del esquema JSON v4 de IETF válido.

Cuando IDT ejecuta su conjunto de pruebas, lee automáticamente el esquema y lo usa para validar el archivo `userdata.json` proporcionado por el ejecutor de la prueba. Si es válido, el contenido del archivo `userdata.json` está disponible tanto en el contexto de IDT como en el contexto del orquestador de pruebas.

Configuración del orquestador de pruebas de IDT

A partir de la versión 4.5.1 de IDT, IDT incluye un nuevo componente orquestador de pruebas. El orquestador de pruebas es un componente de IDT que controla el flujo de ejecución del conjunto de pruebas y genera el informe de prueba una vez que IDT termina de ejecutar todas las pruebas. El orquestador de pruebas determina la selección de las pruebas y el orden en que se ejecutan en función de las reglas definidas por el usuario.

Si su conjunto de pruebas no incluye un orquestador de pruebas definido por el usuario, IDT lo generará.

El orquestador de pruebas predeterminado realiza las siguientes funciones:

- Ofrece a los ejecutores de pruebas la posibilidad de seleccionar y ejecutar grupos de pruebas específicos, en lugar de todo el conjunto de pruebas.
- Si no se seleccionan grupos de pruebas específicos, ejecuta todos los grupos de pruebas del conjunto de pruebas con asignación al azar.
- Genera informes e imprime un resumen de la consola que muestra los resultados de las pruebas de cada grupo y caso de prueba.

El orquestador de pruebas reemplaza al orquestador de pruebas de IDT. Le recomendamos utilizar el orquestador de pruebas para desarrollar sus conjuntos de pruebas en lugar del orquestador de pruebas de IDT. El orquestador de pruebas ofrece las siguientes características mejoradas:

- Utiliza un formato declarativo en comparación con el formato imperativo que utiliza la máquina de estados de IDT. Esto le permite especificar qué pruebas desea ejecutar y cuándo quiere ejecutarlas.
- Administra la gestión de grupos específicos, la generación de informes, la gestión de errores y el seguimiento de los resultados para que no tenga que gestionar estas acciones manualmente.

- Utiliza el formato YAML, que admite comentarios de forma predeterminada.
- Requiere un 80 por ciento menos de espacio en disco que el orquestador de pruebas para definir el mismo flujo de trabajo.
- Añade una validación previa a las pruebas para comprobar que la definición del flujo de trabajo no contiene identificadores de prueba incorrectos o dependencias circulares.

Formato del orquestador de pruebas

Puede utilizar la siguiente plantilla para configurar su propio archivo *<custom-test-suite-folder>*`/suite/test_orchestrator.yaml`:

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Aliases`

Opcional. Cadenas definidas por el usuario que se asignan a expresiones de contexto. Los alias le permiten generar nombres descriptivos para identificar las expresiones de contexto en

la configuración de su orquestador de pruebas. Esto resulta especialmente útil si está creando expresiones contextuales complejas o expresiones que utiliza en varios lugares.

Puede usar expresiones de contexto para almacenar consultas de contexto que le permitan acceder a los datos de otras configuraciones de IDT. Para obtener más información, consulte Acceda a los datos en el contexto.

Example Ejemplo

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Opcional. Una lista de condiciones y los casos de prueba correspondientes que se ejecutan cuando se cumple cada condición. Cada condición puede tener varios casos de prueba; sin embargo, puede asignar un caso de prueba determinado a una sola condición.

De forma predeterminada, IDT ejecuta cualquier caso de prueba que no esté asignado a una condición de esta lista. Si no especifica esta sección, IDT ejecuta todos los grupos de pruebas del conjunto de pruebas.

Cada elemento de la lista `ConditionalTests` incluye los siguientes parámetros:

`Condition`

Una expresión de contexto que se evalúa como un valor booleano. Si el valor evaluado es verdadero, IDT ejecuta los casos de prueba que se especifican en el parámetro `Tests`.

`Tests`

La lista de descriptores de prueba.

Cada descriptor de prueba usa el ID del grupo de pruebas y uno o más ID de casos de prueba para identificar las pruebas individuales que se van a ejecutar en un grupo de pruebas específico. El descriptor de la prueba utiliza el siguiente formato:

```
GroupId: group-id
```

```
CaseIds: [test-id, test-id] # optional
```

Example Ejemplo

En el siguiente ejemplo se utilizan expresiones de contexto genéricas que puede definir como `Aliases`.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

En función de las condiciones definidas, IDT selecciona los grupos de prueba de la siguiente manera:

- Si `Condition1` es verdadero, IDT ejecuta las pruebas de los grupos de pruebas A, B y C.
- Si `Condition2` es verdadero, IDT ejecuta las pruebas de los grupos de pruebas C y D.

`Order`

Opcional. El orden en que se deben ejecutar las pruebas. El orden de las pruebas se especifica a nivel de grupo de pruebas. Si no especifica esta sección, IDT ejecuta todos los grupos de pruebas aplicables en un orden aleatorio. El valor de `Order` es una lista de listas de descriptores de grupos. Cualquier grupo de pruebas que no incluya en la lista `Order` se puede ejecutar en paralelo con cualquier otro grupo de pruebas de la lista.

Cada lista de descriptores de grupo contiene uno o más descriptores de grupo e identifica el orden en el que se deben ejecutar los grupos que se especifican en cada descriptor. Puede utilizar los siguientes formatos para definir descriptores de grupos individuales:

- *`group-id`*: el ID de grupo de un grupo de pruebas existente.
- [*`group-id`*, *`group-id`*]: lista de grupos de pruebas que se pueden ejecutar en cualquier orden relativo.

- `"*"`: comodín. Esto equivale a la lista de todos los grupos de pruebas que aún no están especificados en la lista de descriptores de grupos actual.

El valor de `Order` también debe cumplir los siguientes requisitos:

- Los ID de los grupos de pruebas que especifique en un descriptor de grupo deben existir en su conjunto de pruebas.
- Cada lista de descriptores de grupos debe incluir al menos un grupo de pruebas.
- Cada lista de descriptores de grupo debe contener identificadores de grupo únicos. No puede repetir el ID de un grupo de pruebas dentro de los descriptores de grupos individuales.
- Una lista de descriptores de grupo puede tener como máximo un descriptor de grupo comodín. El descriptor de grupo comodín debe ser el primer o el último elemento de la lista.

Example Ejemplos

En el caso de un conjunto de pruebas que contiene los grupos de pruebas A, B, C, D y E, en la siguiente lista de ejemplos se muestran diferentes formas de especificar que IDT debe ejecutar primero el grupo de pruebas A, después el grupo de pruebas B y, por último, ejecutar los grupos de pruebas C, D y E en cualquier orden.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
  ```
  ```

```
      - E
```

`Features`

Opcional. La lista de características del producto que desea que IDT añada al archivo `awsiotdevicetester_report.xml`. Si no especifica esta sección, IDT no añadirá ninguna característica del producto al informe.

Una característica del producto es información definida por el usuario sobre los criterios específicos que puede cumplir un dispositivo. Por ejemplo, la característica MQTT del producto puede indicar que el dispositivo publica los mensajes MQTT correctamente. En `awsiotdevicetester_report.xml`, las características del producto se establecen como `supported`, `not-supported` o como un valor personalizado, en función de si se han superado las pruebas especificadas.

Cada elemento de la lista `Features` incluye los siguientes parámetros:

`Name`

El nombre de la característica.

`Value`

Opcional. El valor personalizado que desea utilizar en el informe en lugar de `supported`. Si no se especifica este valor, IDT establece el valor de la característica en `supported` o `not-supported`, en función de los resultados de las pruebas. Si prueba la misma característica con diferentes condiciones, puede utilizar un valor personalizado para cada instancia de esa característica en la lista `Features` e IDT concatena los valores de la característica para las condiciones admitidas. Para más información, consulte

`Condition`

Una expresión de contexto que se evalúa como un valor booleano. Si el valor evaluado es verdadero, IDT añade la característica al informe de la prueba una vez que termine de ejecutar el conjunto de pruebas. Si el valor evaluado es falso, la prueba no se incluye en el informe.

`Tests`

Opcional. La lista de descriptores de prueba. Para que la característica sea compatible, se deben superar todas las pruebas especificadas en esta lista.

Cada descriptor de prueba de esta lista usa el ID del grupo de pruebas y uno o más ID de casos de prueba para identificar las pruebas individuales que se van a ejecutar en un grupo de pruebas específico. El descriptor de la prueba utiliza el siguiente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Debe especificar `Tests` o `OneOfTests` para cada característica de la lista `Features`.

`OneOfTests`

Opcional. La lista de descriptores de prueba. Para que la característica sea compatible, se debe superar al menos una de las pruebas especificadas en esta lista.

Cada descriptor de prueba de esta lista usa el ID del grupo de pruebas y uno o más ID de casos de prueba para identificar las pruebas individuales que se van a ejecutar en un grupo de pruebas específico. El descriptor de la prueba utiliza el siguiente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Debe especificar `Tests` o `OneOfTests` para cada característica de la lista `Features`.

`IsRequired`

El valor booleano que define si la característica es obligatoria en el informe de prueba. El valor predeterminado es `false`.

Example

Contexto del orquestador de pruebas

El contexto del orquestador de pruebas es un documento JSON de solo lectura que contiene datos que están disponibles para el orquestador de pruebas durante la ejecución. Solo se puede acceder al contexto del orquestador de pruebas desde el orquestador de pruebas y contiene información que determina el flujo de prueba. Por ejemplo, puede usar la información configurada por los ejecutores de la prueba en el archivo `userdata.json` para determinar si es necesario ejecutar una prueba específica.

El contexto del orquestador de pruebas utiliza el siguiente formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

Información sobre el grupo de dispositivos seleccionado para la ejecución de la prueba. Para un grupo de dispositivos seleccionados, esta información se recupera del elemento correspondiente de la matriz del grupo de dispositivos de alto nivel definido en el archivo `device.json`.

`userData`

Información en el archivo `userdata.json`.

`config`

Información en el archivo `config.json`.

Puede consultar el contexto mediante la notación JSONPath. La sintaxis de las consultas JSONPath en las definiciones de estado es `{{query}}`. Al acceder a los datos desde el contexto del orquestador de pruebas, asegúrese de que cada valor se evalúe como una cadena, un número o un booleano.

Para obtener más información sobre cómo utilizar la notación JSONPath para acceder a los datos desde el contexto, consulte Uso del contexto de IDT.

Configure la máquina de estados IDT

> **Important**
>
> A partir de la versión 4.5.1 de IDT, esta máquina de estados está obsoleta. Le recomendamos encarecidamente que utilice el nuevo orquestador de pruebas. Para obtener más información, consulte Configuración del orquestador de pruebas de IDT.

Una máquina de estados es un constructo que controla el flujo de ejecución del conjunto de pruebas. Determina el estado inicial de un conjunto de pruebas, administra las transiciones de estado en función de las reglas definidas por el usuario y continúa pasando por esos estados hasta alcanzar el estado final.

Si su conjunto de pruebas no incluye una máquina de estados definida por el usuario, IDT la generará. La máquina de estados predeterminada realiza las siguientes funciones:

- Ofrece a los ejecutores de pruebas la posibilidad de seleccionar y ejecutar grupos de pruebas específicos, en lugar de todo el conjunto de pruebas.
- Si no se seleccionan grupos de pruebas específicos, ejecuta todos los grupos de pruebas del conjunto de pruebas con asignación al azar.
- Genera informes e imprime un resumen de la consola que muestra los resultados de las pruebas de cada grupo y caso de prueba.

La máquina de estados de un conjunto de pruebas de IDT debe cumplir los siguientes criterios:

- Cada estado corresponde a una acción que debe realizar IDT, como ejecutar un grupo de pruebas o crear un archivo de informe.
- La transición a un estado ejecuta la acción asociada a ese estado.
- Cada estado define la regla de transición para el siguiente estado.
- El estado final debe ser `Succeed` o `Fail`.

Formato de las máquinas de estados

Puede utilizar la siguiente plantilla para configurar su propio archivo *<custom-test-suite-folder>*`/suite/state_machine.json`:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
```

```
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Comment`

Una descripción de la máquina de estados.

`StartAt`

El nombre del estado en el que IDT comienza a ejecutar el conjunto de pruebas. El valor de `StartAt` debe estar establecido en uno de los estados enumerados en el objeto `States`.

`States`

Objeto que asigna los nombres de estado definidos por el usuario a estados de IDT válidos. Cada uno de los estados. *`state-name`*el objeto contiene la definición de un estado válido asignado a. *`state-name`*

El objeto `States` debe incluir los estados `Succeed` y `Fail`. Para obtener información sobre los estados válidos, consulte Estados válidos y definiciones de estado.

Estados válidos y definiciones de estado

En esta sección se describen las definiciones de estado de todos los estados válidos que se pueden usar en la máquina de estados de IDT. Algunos de los siguientes estados admiten configuraciones en el nivel de caso de prueba. Sin embargo, le recomendamos que configure las reglas de transición de estado en el nivel de grupo de pruebas en lugar de en el nivel del caso de prueba, a menos que sea absolutamente necesario.

Definiciones de estado

- RunTask
- Choice
- Parallel

- AddProductFeatures
- Informar
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

El estado `RunTask` ejecuta casos de prueba a partir de un grupo de pruebas definido en el conjunto de pruebas.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`TestGroup`

Opcional. El ID del grupo de pruebas que se va a ejecutar. Si no se especifica este valor, IDT ejecuta el grupo de pruebas que seleccione el ejecutor de la prueba.

`TestCases`

Opcional. Una matriz de casos de prueba IDs del grupo especificado en`TestGroup`. En función de los valores de `TestGroup` y `TestCases`, IDT determina el comportamiento de la ejecución de la prueba de la siguiente manera:

- Cuando se especifica `TestGroup` y `TestCases`, IDT ejecuta los casos de prueba especificados del grupo de pruebas.
- Cuando se especifica `TestCases`, pero no se especifica `TestGroup`, IDT ejecuta los casos de prueba especificados.
- Cuando se especifica `TestGroup`, pero no se especifica `TestCases`, IDT ejecuta todos los casos de prueba del grupo de pruebas especificado.
- Si no se especifica `TestGroup` ni `TestCases`, IDT ejecuta todos los casos de prueba del grupo de pruebas que el ejecutor de la prueba selecciona en la CLI de IDT. Para habilitar la selección de grupos para los ejecutores de las pruebas, debe incluir los estados `RunTask` y `Choice` en el archivo `state_machine.json`. Para ver un ejemplo de cómo funciona, consulte Ejemplo de máquina de estados: ejecutar grupos de prueba seleccionados por el usuario.

  Para obtener más información sobre cómo habilitar los comandos CLI de IDT para los ejecutores de pruebas, consulte the section called “Habilitación de comandos de CLI de IDT”.

`ResultVar`

El nombre de la variable de contexto que se va a configurar con los resultados de la prueba. No especifique este valor si no especificó ningún valor para `TestGroup`. IDT establece el valor de la variable que defina en `ResultVar` como `true` o `false` en función de lo siguiente:

- Si el nombre de la variable tiene el formato *`text`*`_`*`text`*`_passed`, el valor se establece en función de si todas las pruebas del primer grupo de pruebas se aprobaron o se omitieron.
- En todos los demás casos, el valor se establece en función de si todas las pruebas de todos los grupos de pruebas se aprobaron o se omitieron.

Normalmente, se utiliza el `RunTask` estado para especificar un ID de grupo de pruebas sin especificar un caso de prueba individual IDs, de modo que IDT ejecutará todos los casos de prueba del grupo de pruebas especificado. Todos los casos de prueba ejecutados por este estado se ejecutan en paralelo, en orden aleatorio. Sin embargo, si todos los casos de prueba requieren la ejecución de un dispositivo y solo hay un dispositivo disponible, los casos de prueba se ejecutarán secuencialmente.

Error handling (Control de errores)

Si alguno de los grupos de pruebas o casos IDs de prueba especificados no es válido, este estado genera el error de `RunTaskError` ejecución. Si el estado encuentra un error de ejecución, también establece la variable `hasExecutionError` en el contexto de la máquina de estados en `true`.

Choice

El estado `Choice` le permite configurar dinámicamente el siguiente estado al que realizar la transición en función de las condiciones definidas por el usuario.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Default`

El estado predeterminado al que se realizará la transición si no se puede evaluar ninguna de las expresiones definidas en `Choices` como `true`.

`FallthroughOnError`

Opcional. Especifica el comportamiento cuando el estado encuentra un error al evaluar las expresiones. Establézcalo en `true` si desea omitir una expresión si la evaluación genera un error. Si ninguna expresión coincide, la máquina de estados pasa al estado `Default`. Si no se especifica el valor `FallthroughOnError`, se establece de forma predeterminada en `false`.

`Choices`

Una matriz de expresiones y estados para determinar a qué estado hacer la transición después de ejecutar las acciones en el estado actual.

`Choices.Expression`

Una cadena de expresión que se evalúa como un valor booleano. Si la expresión se evalúa como `true`, la máquina de estados pasa al estado definido en `Choices.Next`. Las cadenas de expresión recuperan los valores del contexto de la máquina de estados y, a continuación, realizan operaciones en ellos para obtener un valor booleano. Para obtener información sobre

cómo acceder al contexto de la máquina de estados, consulte Contexto de la máquina de estados.

`Choices.Next`

El nombre del estado al que se realizará la transición si la expresión definida en `Choices.Expression` se evalúa como `true`.

Error handling (Control de errores)

El estado `Choice` puede requerir la gestión de errores en los siguientes casos:

- Algunas variables de las expresiones de elección no existen en el contexto de la máquina de estados.
- El resultado de una expresión no es un valor booleano.
- El resultado de una búsqueda en JSON no es una cadena, un número ni un booleano.

No puede usar un bloque `Catch` para gestionar los errores en este estado. Si quiere detener la ejecución de la máquina de estados cuando encuentre un error, debe establecer `FallthroughOnError` en `false`. Sin embargo, le recomendamos que establezca `FallthroughOnError` en `true` y, en función de su caso de uso, haga una de las siguientes opciones:

- Si se espera que una variable a la que está accediendo no exista en algunos casos, utilice el valor `Default` y los bloques `Choices` adicionales para especificar el siguiente estado.
- Si una variable a la que está accediendo debe existir siempre, defina el estado `Default` en `Fail`.

Parallel

El estado `Parallel` le permite definir y ejecutar nuevas máquinas de estados en paralelo entre sí.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`Branches`

Una matriz de definiciones de máquinas de estados para ejecutar. Cada definición de máquina de estados debe contener sus propios estados `StartAt`, `Succeed` y `Fail`. Las definiciones de máquinas de estados de esta matriz no pueden hacer referencia a estados ajenos a su propia definición.

> **Note**
>
> Como cada máquina de estados de la rama comparte el mismo contexto de máquina de estados, establecer variables en una rama y, a continuación, leer esas variables desde otra rama podría provocar un comportamiento inesperado.

El estado `Parallel` pasa al siguiente estado solo después de ejecutar todas las máquinas de estados de rama. Cada estado que requiera un dispositivo esperará para ejecutarse hasta que el dispositivo esté disponible. Si hay varios dispositivos disponibles, este estado ejecuta casos de prueba de varios grupos en paralelo. Si no hay suficientes dispositivos disponibles, los casos de prueba se ejecutarán secuencialmente. Como los casos de prueba se ejecutan en orden aleatorio cuando se ejecutan en paralelo, se podrían usar diferentes dispositivos para ejecutar pruebas del mismo grupo de pruebas.

Error handling (Control de errores)

Asegúrese de que tanto la máquina de estados de la rama como la máquina de estados principal pasen al estado `Fail` para gestionar los errores de ejecución.

Como las máquinas de estados de la rama no transmiten los errores de ejecución a la máquina de estados principal, no puede usar un bloque `Catch` para gestionar los errores de ejecución en las máquinas de estados de la rama. En su lugar, utilice el valor `hasExecutionErrors` en el contexto de la máquina de estados compartida. Para ver un ejemplo de cómo funciona, consulte Ejemplo de máquina de estados: ejecutar dos grupos de prueba en paralelo.

AddProductFeatures

El estado `AddProductFeatures` le permite añadir características del producto al archivo `awsiotdevicetester_report.xml` generado por IDT.

Una característica del producto es información definida por el usuario sobre los criterios específicos que puede cumplir un dispositivo. Por ejemplo, la característica del producto `MQTT` puede indicar que el dispositivo publica los mensajes MQTT correctamente. En el informe, las características del producto se establecen como `supported`, `not-supported` o como un valor personalizado, en función de si se han superado las pruebas especificadas.

> **Note**
>
> El estado `AddProductFeatures` no genera informes por sí mismo. Este estado debe realizar la transición al estado `Report` para generar informes.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`Features`

Una matriz de características del producto para mostrar en el archivo `awsiotdevicetester_report.xml`.

`Feature`

El nombre de la característica

`FeatureValue`

Opcional. El valor personalizado que se utilizará en el informe en lugar de `supported`. Si no se especifica este valor, según los resultados de las pruebas, el valor de la característica se establece en `supported` o `not-supported`.

Si utiliza un valor personalizado para `FeatureValue`, puede probar la misma característica con diferentes condiciones e IDT concatena los valores de la característica para las condiciones admitidas. Por ejemplo, en el siguiente fragmento se muestra la característica `MyFeature` con dos valores de característica distintos:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Si ambos grupos de pruebas aprueban, el valor de la característica se establece en `first-feature-supported, second-feature-supported`.

`Groups`

Opcional. Matriz de grupos de prueba IDs. Para que la característica sea compatible, deben superan la prueba todas las pruebas de cada grupo de pruebas especificado.

`OneOfGroups`

Opcional. Una matriz de grupos de prueba IDs. Para que la característica sea compatible, deben aprobarse todas las pruebas de al menos uno de los grupos de pruebas especificados.

`TestCases`

Opcional. Una serie de casos de prueba IDs. Si especifica este valor, se aplicará lo siguiente:

- Para que la característica sea compatible, deben superan la prueba todos los casos de prueba especificados.
- `Groups` debe contener solo un ID de grupo de pruebas.
- `OneOfGroups` no debe especificarse.

`IsRequired`

Opcional. Establézcalo en `false` para marcar esta característica como una característica opcional en el informe. El valor predeterminado es `true`.

`ExecutionMethods`

Opcional. Una matriz de métodos de ejecución que coinciden con el valor `protocol` especificado en el archivo `device.json`. Si se especifica este valor, los ejecutores de pruebas deben especificar un valor `protocol` que coincida con uno de los valores de esta matriz para incluir la característica en el informe. Si no se especifica este valor, la característica siempre se incluirá en el informe.

Para usar el estado `AddProductFeatures`, debe establecer el valor de `ResultVar` con estado `RunTask` en uno de los siguientes valores:

- Si especificó un caso de prueba individual IDs, `ResultVar` configúrelo en*`group-id_test-id`*`_passed`.
- Si no especificó un caso de prueba individual IDs, `ResultVar` configúrelo en*`group-id`*`_passed`.

El estado `AddProductFeatures` comprueba los resultados de las pruebas de la siguiente manera:

- Si no especificó ningún caso de prueba IDs, el resultado de cada grupo de prueba se determina a partir del valor de la *group-id*_passed variable en el contexto de la máquina de estados.
- Si especificó un caso de prueba IDs, el resultado de cada una de las pruebas se determina a partir del valor de la *group-id_test-id*_passed variable en el contexto de la máquina de estados.

Error handling (Control de errores)

Si un identificador de grupo proporcionado en este estado no es un identificador de grupo válido, este estado provoca un error de ejecución de `AddProductFeaturesError`. Si el estado encuentra un error de ejecución, también establece la variable `hasExecutionErrors` en el contexto de la máquina de estados en `true`.

Informar

El estado `Report` genera los archivos *suite-name*`_Report.xml` y `awsiotdevicetester_report.xml`. Este estado también transmite el informe a la consola.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

Siempre debe pasar al estado `Report` que se encuentra al final del flujo de ejecución de la prueba para que los ejecutores de la prueba puedan ver los resultados de la prueba. Normalmente, el siguiente estado después de este estado es `Succeed`.

Error handling (Control de errores)

Si este estado tiene problemas con la generación de los informes, se produce el error de ejecución `ReportError`.

LogMessage

El estado `LogMessage` genera el archivo `test_manager.log` y transmite el mensaje de registro a la consola.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`Level`

El nivel de error en el que se va a crear el mensaje de registro. Si especifica un nivel que no es válido, este estado genera un mensaje de error y lo descarta.

`Message`

El mensaje para registrar.

SelectGroup

El estado `SelectGroup` actualiza el contexto de la máquina de estados para indicar qué grupos están seleccionados. Los valores establecidos por este estado se utilizan en cualquier estado `Choice` posterior.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Next`

El nombre del estado al que se realizará la transición después de ejecutar las acciones en el estado actual.

`TestGroups`

Una matriz de grupos de pruebas que se marcarán como seleccionados. Para cada ID de grupo de pruebas de esta matriz, la variable *`group-id`*`_selected` se establece `true` en el contexto. Asegúrese de proporcionar un grupo de prueba válido, IDs ya que IDT no valida la existencia de los grupos especificados.

Fail

El estado `Fail` indica que la máquina de estados no se ejecutó correctamente. Este es el estado final de la máquina de estados y cada definición de la máquina de estados debe incluir este estado.

```
{
    "Type": "Fail"
}
```

Succeed

El estado `Succeed` indica que la máquina de estados se ejecutó correctamente. Este es el estado final de la máquina de estados y cada definición de la máquina de estados debe incluir este estado.

```
{
    "Type": "Succeed"
}
```

Contexto de la máquina de estados

El contexto de la máquina de estados es un documento JSON de solo lectura que contiene datos que están disponibles para la máquina de estados durante la ejecución. Solo se puede acceder al contexto de la máquina de estados desde la máquina de estados y contiene información que determina el flujo de prueba. Por ejemplo, puede usar la información configurada por los ejecutores de la prueba en el archivo `userdata.json` para determinar si es necesario ejecutar una prueba específica.

El contexto de la máquina de estados usa el siguiente formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

`pool`

Información sobre el grupo de dispositivos seleccionado para la ejecución de la prueba. Para un grupo de dispositivos seleccionados, esta información se recupera del elemento correspondiente de la matriz del grupo de dispositivos de alto nivel definido en el archivo `device.json`.

`userData`

Información en el archivo `userdata.json`.

`config`

Información del archivo `config.json`.

`suiteFailed`

El valor se establece en `false` cuando se inicia la máquina de estados. Si un grupo de pruebas falla en un estado `RunTask`, este valor se establece en `true` durante el resto de la ejecución de la máquina de estados.

`specificTestGroups`

Si el responsable de la prueba selecciona grupos de pruebas específicos para ejecutarlos en lugar de todo el conjunto de pruebas, se crea esta clave y contiene la lista de grupos IDs de pruebas específicos.

`specificTestCases`

Si el ejecutor de la prueba selecciona casos de prueba específicos para ejecutarlos en lugar de todo el conjunto de pruebas, se crea esta clave y contiene la lista de casos de prueba específicos IDs.

`hasExecutionErrors`

No se cierra cuando se inicia la máquina de estados. Si algún estado detecta errores de ejecución, se crea esta variable y se establece en `true` durante el resto de la ejecución de la máquina de estados.

Puede consultar el contexto mediante la JSONPath notación. La sintaxis de las JSONPath consultas en las definiciones de estado es`{{$.query}}`. Puede utilizar JSONPath las consultas como cadenas de marcadores de posición en algunos estados. IDT reemplaza las cadenas de marcadores de posición por el valor de la JSONPath consulta evaluada en el contexto. Puede utilizar los siguientes marcadores de posición para los siguientes valores:

- El valor `TestCases` en estados `RunTask`.
- El valor `Expression` en estado `Choice`.

Cuando accede a los datos del contexto de la máquina de estados, asegúrese de que se cumplan las siguientes condiciones:

- Sus rutas de JSON deben comenzar por `$.`
- Cada valor debe evaluarse como una cadena, un número o un booleano.

Para obtener más información sobre el uso de la JSONPath notación para acceder a los datos del contexto, consulte. Uso del contexto de IDT

Errores de ejecución

Los errores de ejecución son errores en la definición de la máquina de estados que esta encuentra al ejecutar un estado. IDT registra la información sobre cada error en el archivo `test_manager.log` y transmite el mensaje de registro a la consola.

Puede utilizar los siguientes métodos para gestionar los errores de ejecución:

- Añada un bloque `Catch` a la definición de estado.

- Compruebe el valor del valor `hasExecutionErrors` en el contexto de la máquina de estados.

Catch

Para usar `Catch`, añada lo siguiente a su definición de estado:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`Catch.ErrorEquals`

Una matriz de los tipos de error que se deben capturar. Si un error de ejecución coincide con uno de los valores especificados, la máquina de estados pasa al estado especificado en `Catch.Next`. Consulte la definición de cada estado para obtener información sobre el tipo de error que genera.

`Catch.Next`

El siguiente estado al que se realizará la transición si el estado actual encuentra un error de ejecución que coincide con uno de los valores especificados en `Catch.ErrorEquals`.

Los bloques Catch se gestionan secuencialmente hasta que uno coincide. Si los errores no coinciden con los enumerados en los bloques Catch, las máquinas de estado seguirán ejecutándose. Como los errores de ejecución son el resultado de definiciones de estado incorrectas, se recomienda que pase al estado de Error cuando un estado detecte un error de ejecución.

hasExecutionError

Cuando algunos estados encuentran errores de ejecución, además de emitir el error, también establecen el valor `hasExecutionError` en `true` en el contexto de la máquina de estados. Puede usar este valor para detectar cuándo se produce un error y, a continuación, usar un estado `Choice` para hacer la transición de la máquina de estados al estado `Fail`.

Este método incluye las siguientes características:

- La máquina de estados no comienza con ningún valor asignado a `hasExecutionError` y este valor no está disponible hasta que se establezca en un estado concreto. Esto significa que debe establecer explícitamente `FallthroughOnError` en `false` para los estados `Choice` que acceden a este valor para evitar que la máquina de estados se detenga si no se produce ningún error de ejecución.
- Una vez establecido en `true`, `hasExecutionError` nunca se establece en falso ni se elimina del contexto. Esto significa que este valor solo es útil la primera vez que se establece en `true` y, para todos los estados posteriores, no proporciona un valor significativo.
- El valor `hasExecutionError` se comparte con todas las máquinas de estados de la rama con el estado `Parallel`, lo que puede provocar resultados inesperados en función del orden en que se acceda a él.

Debido a estas características, no recomendamos utilizar este método si se puede utilizar un bloque Catch en su lugar.

Máquinas de estados de ejemplo

En esta sección se proporcionan algunos ejemplos de configuraciones de máquinas de estados.

Ejemplos

- Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas
- Ejemplo de máquina de estados: ejecutar grupos de pruebas seleccionados por el usuario
- Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas con características de productos
- Ejemplo de máquina de estados: ejecutar dos grupos de prueba en paralelo

Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas

Esta máquina de estados:

- Ejecuta el grupo de pruebas con ID `GroupA`, que debe estar presente en el conjunto de un archivo `group.json`.
- Comprueba si hay errores de ejecución y pasa a `Fail` si se encuentra alguno.
- Genera un informe y pasa a `Succeed` si no hay errores y a `Fail` en caso contrario.

```
{
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ejemplo de máquina de estados: ejecutar grupos de pruebas seleccionados por el usuario

Esta máquina de estados:

- Comprueba si el ejecutor de pruebas seleccionó grupos de pruebas específicos. La máquina de estados no comprueba si hay casos de prueba específicos porque los ejecutores de pruebas no pueden seleccionar casos de prueba sin seleccionar también un grupo de pruebas.
- Si se seleccionan grupos de pruebas:
  - Ejecuta los casos de prueba dentro de los grupos de pruebas seleccionados. Para ello, la máquina de estados no especifica explícitamente ningún grupo de pruebas o casos de prueba en el estado `RunTask`.
  - Genera un informe después de ejecutar todas las pruebas y sale.
- Si no se seleccionan grupos de pruebas:
  - Ejecuta las pruebas del grupo de pruebas `GroupA`.
  - Genera informes y sale.

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
                    "Next": "RunSpecificGroups"
                }
            ]
        },
        "RunSpecificGroups": {
            "Type": "RunTask",
            "Next": "Report",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
```

```
        },
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ejemplo de máquina de estados: ejecutar un solo grupo de pruebas con características de productos

Esta máquina de estados:

- Ejecuta el grupo de pruebas `GroupA`.
- Comprueba si hay errores de ejecución y pasa a `Fail` si se encuentra alguno.
- Añade la característica `FeatureThatDependsOnGroupA` al archivo `awsiotdevicetester_report.xml`:

  - Si `GroupA` supera la prueba, la característica se establece en `supported`.
  - La característica no se marca como opcional en el informe.
- Genera un informe y pasa a `Succeed` si no hay errores y a `Fail` en caso contrario.

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
```

```
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Ejemplo de máquina de estados: ejecutar dos grupos de prueba en paralelo

Esta máquina de estados:

- Ejecuta los grupos de pruebas `GroupA` y `GroupB` en paralelo. Las variables `ResultVar` almacenadas en el contexto por los estados `RunTask` de las máquinas de estados de la rama están disponibles para el estado `AddProductFeatures`.
- Comprueba si hay errores de ejecución y pasa a `Fail` si se encuentra alguno. Esta máquina de estados no utiliza un bloque `Catch` porque ese método no detecta errores de ejecución en las máquinas de estados de la rama.
- Agrega características al archivo `awsiotdevicetester_report.xml` en función de los grupos que aprueban
  - Si `GroupA` supera la prueba, la característica se establece en `supported`.
  - La característica no se marca como opcional en el informe.
- Genera un informe y pasa a `Succeed` si no hay errores y a `Fail` en caso contrario.

Si hay dos dispositivos configurados en el grupo de dispositivos, ambos `GroupA` y `GroupB` pueden ejecutarse al mismo tiempo. Sin embargo, si `GroupA` o `GroupB` incluyen varias pruebas, es posible que ambos dispositivos se asignen a esas pruebas. Si solo se configura un dispositivo, los grupos de prueba se ejecutarán secuencialmente.

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
```

```
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
                            "Next": "Succeed",
                            "TestGroup": "GroupA",
                            "ResultVar": "GroupA_passed",
                            "Catch": [
                                {
                                    "ErrorEquals": [
                                        "RunTaskError"
                                    ],
                                    "Next": "Fail"
                                }
                            ]
                        },
                        "Succeed": {
                            "Type": "Succeed"
                        },
                        "Fail": {
                            "Type": "Fail"
                        }
                    }
                },
                {
                    "Comment": "Run GroupB state machine",
                    "StartAt": "RunGroupB",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
                            "Next": "Succeed",
                            "TestGroup": "GroupB",
                            "ResultVar": "GroupB_passed",
                            "Catch": [
                                {
                                    "ErrorEquals": [
                                        "RunTaskError"
```

```
                                ],
                                "Next": "Fail"
                            }
                        ]
                    },
                    "Succeed": {
                        "Type": "Succeed"
                    },
                    "Fail": {
                        "Type": "Fail"
                    }
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
            }
        ]
    },
    "AddProductFeatures": {
        "Type": "AddProductFeatures",
        "Next": "Report",
        "Features": [
            {
                "Feature": "FeatureThatDependsOnGroupA",
                "Groups": [
                    "GroupA"
                ],
                "IsRequired": true
            },
            {
                "Feature": "FeatureThatDependsOnGroupB",
                "Groups": [
                    "GroupB"
                ],
                "IsRequired": true
            }
```

```
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Cree ejecutables de casos de prueba de IDT

Puede crear y colocar ejecutables de casos de prueba en una carpeta de conjunto de pruebas de las siguientes maneras:

- En el caso de los conjuntos de pruebas que utilizan argumentos o variables de entorno de los archivos `test.json` para determinar qué pruebas se van a ejecutar, puede crear un único ejecutable de caso de prueba para todo el conjunto de pruebas o un ejecutable de prueba para cada grupo de pruebas del conjunto de pruebas.
- En el caso de un conjunto de pruebas en el que desee ejecutar pruebas específicas en función de comandos específicos, debe crear un ejecutable de caso de prueba para cada caso de prueba del conjunto de pruebas.

Como redactor de pruebas, puede determinar qué enfoque es adecuado para su caso de uso y estructurar el ejecutable del caso de prueba en consecuencia. Asegúrese de proporcionar la ruta correcta al ejecutable del caso de prueba en cada archivo `test.json` y de que el ejecutable especificado se ejecute correctamente.

Cuando todos los dispositivos estén preparados para ejecutar un caso de prueba, IDT lee los siguientes archivos:

- El `test.json` para el caso de prueba seleccionado determina los procesos que se van a iniciar y las variables de entorno que se van a configurar.
- El `suite.json` para el conjunto de pruebas determina las variables de entorno que se van a configurar.

IDT inicia el proceso del ejecutable de prueba requerido en función de los comandos y argumentos especificados en el archivo `test.json` y pasa las variables de entorno requeridas al proceso.

Uso del SDK de cliente de IDT

Los SDK de cliente de IDT le permiten simplificar la forma de escribir la lógica de prueba en su ejecutable de prueba con comandos de API que puede utilizar para interactuar con IDT y los dispositivos que se están probando. Actualmente, IDT ofrece los siguientes SDK:

- SDK de cliente de IDT para Python
- SDK de cliente de IDT para Go
- SDK de cliente de IDT para Java

Estos SDK se encuentran en la carpeta *<device-tester-extract-location>*`/sdks`. Al crear un ejecutable de caso de prueba nuevo, debe copiar el SDK que quiere usar en la carpeta que contiene el ejecutable del caso de prueba y hacer referencia al SDK en su código. En esta sección se proporciona una breve descripción de los comandos de API disponibles que puede usar en los ejecutables de casos de prueba.

En esta sección

- Interacción con el dispositivo
- Interacción con IDT
- Interacción con el host

Interacción con el dispositivo

Los siguientes comandos le permiten comunicarse con el dispositivo que se está probando sin tener que implementar ninguna función adicional de administración de la conectividad e interacción del dispositivo.

`ExecuteOnDevice`

Permite que los conjuntos de pruebas ejecuten intérpretes de comandos en un dispositivo que admite conexiones de intérpretes de comandos SSH o Docker.

`CopyToDevice`

Permite a los conjuntos de pruebas copiar un archivo local desde la máquina host que ejecuta IDT a una ubicación específica de un dispositivo que admite conexiones de intérpretes de comandos SSH o Docker.

`ReadFromDevice`

Permite que los conjuntos de pruebas lean desde el puerto de serie de los dispositivos que admiten conexiones UART.

> **Note**
>
> Dado que IDT no gestiona las conexiones directas a los dispositivos que se realizan con información de acceso a los dispositivos procedente del contexto, recomendamos utilizar estos comandos de la API de interacción del dispositivo en los ejecutables de casos de prueba. Sin embargo, si estos comandos no cumplen los requisitos del caso de prueba, puede recuperar la información de acceso al dispositivo desde el contexto de IDT y utilizarla para establecer una conexión directa con el dispositivo desde el conjunto de pruebas. Para establecer una conexión directa, recupere la información de los campos `device.connectivity` y `resource.devices.connectivity` del dispositivo que se está probando y de los dispositivos de recursos, respectivamente. Para obtener más información sobre cómo usar el contexto de IDT, consulte Uso del contexto de IDT.

Interacción con IDT

Los siguientes comandos permiten que sus conjuntos de pruebas se comuniquen con IDT.

`PollForNotifications`

Permite que los conjuntos de pruebas comprueben las notificaciones de IDT.

`GetContextValue` y `GetContextString`

Permite que los conjuntos de pruebas recuperen valores del contexto de IDT. Para obtener más información, consulte Uso del contexto de IDT.

`SendResult`

Permite que los conjuntos de pruebas notifiquen los resultados de los casos de prueba a IDT. Debe llamarse a este comando al final de cada caso de prueba en un conjunto de pruebas.

Interacción con el host

El siguiente comando permite que sus conjuntos de pruebas se comuniquen con la máquina host.

`PollForNotifications`

Permite que los conjuntos de pruebas comprueben las notificaciones de IDT.

`GetContextValue` y `GetContextString`

Permite que los conjuntos de pruebas recuperen valores del contexto de IDT. Para obtener más información, consulte Uso del contexto de IDT.

`ExecuteOnHost`

Permite que los conjuntos de pruebas ejecuten comandos en la máquina local y permite a IDT gestionar el ciclo de vida de los casos de prueba ejecutables.

Habilitación de comandos de CLI de IDT

El comando `run-suite` de la CLI de IDT proporciona varias opciones que permiten al ejecutor de pruebas personalizar la ejecución de las pruebas. Para permitir que los ejecutores de pruebas utilicen estas opciones para ejecutar su conjunto de pruebas personalizado, debe implementar la compatibilidad con la CLI de IDT. Si no implementa la compatibilidad, los ejecutores de pruebas podrán seguir ejecutándolas, pero algunas opciones de CLI no funcionarán correctamente. Para ofrecer una experiencia de cliente ideal, le recomendamos que implemente la compatibilidad con los siguientes argumentos para el comando `run-suite` en la CLI de IDT:

`timeout-multiplier`

Especifica un valor superior a 1,0 que se aplicará a todos los tiempos de espera durante la ejecución de las pruebas.

Los ejecutores de pruebas pueden usar este argumento para aumentar el tiempo de espera de los casos de prueba que desean ejecutar. Cuando un ejecutor de pruebas especifica este argumento en su comando `run-suite`, IDT lo usa para calcular el valor de la variable de entorno IDT_TEST_TIMEOUT y establece el campo `config.timeoutMultiplier` en el contexto de IDT. Para que se admita este argumento, debe hacer lo siguiente:

- En lugar de utilizar directamente el valor de tiempo de espera del archivo `test.json`, lea la variable de entorno IDT_TEST_TIMEOUT para obtener el valor de tiempo de espera calculado correctamente.
- Recupere el valor `config.timeoutMultiplier` del contexto de IDT y aplíquelo a los tiempos de espera prolongados.

Para obtener más información sobre cómo salir anticipadamente debido a eventos de tiempo de espera, consulte Especificación del comportamiento de salida.

`stop-on-first-failure`

Especifica que IDT debe dejar de ejecutar todas las pruebas si detecta un error.

Cuando un ejecutor de pruebas especifica este argumento en su comando `run-suite`, IDT dejará de ejecutar las pruebas en cuanto detecte un error. Sin embargo, si los casos de prueba se ejecutan en paralelo, esto puede generar resultados inesperados. Para implementar la compatibilidad, asegúrese de que si IDT detecta este evento, su lógica de pruebas indique a todos los casos de prueba en ejecución que se detengan, se limpien los recursos temporales y se notifique el resultado de la prueba a IDT. Para obtener más información sobre cómo salir anticipadamente en caso de error, consulte Especificación del comportamiento de salida.

`group-id` y `test-id`

Especifica que IDT debe ejecutar solo los grupos de pruebas o los casos de prueba seleccionados.

Los ejecutores de pruebas pueden usar estos argumentos con su `run-suite` comando para especificar el siguiente comportamiento de ejecución de la prueba:

- Ejecutar todas las pruebas dentro de los grupos de pruebas especificados.
- Ejecutar una selección de pruebas desde un grupo de pruebas especificado.

Para admitir estos argumentos, el orquestador de prueba de su conjunto de pruebas debe incluir un conjunto específico de estados `RunTask` y `Choice` en su orquestador de prueba. Si no usa una máquina de estados personalizada, el orquestador de prueba IDT predeterminado incluye los estados necesarios y no es necesario que realice ninguna acción adicional. Sin embargo, si usa un orquestador de prueba personalizado, use Ejemplo de máquina de estados: ejecutar grupos de pruebas seleccionados por el usuario como ejemplo para agregar los estados necesarios a su orquestador de prueba.

Para obtener más información sobre los comandos de la CLI de IDT, consulte Depurar y ejecutar conjuntos de pruebas personalizadas.

Escritura de registros de eventos

Mientras se ejecuta la prueba, se envían datos a `stdout` y `stderr` para escribir registros de eventos y mensajes de error en la consola. Para obtener más información sobre el formato de los mensajes de la consola, consulte Formato de mensajes de consola.

Cuando IDT termina de ejecutar el conjunto de pruebas, esta información también está disponible en el archivo `test_manager.log` ubicado en la carpeta *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs`.

Puede configurar cada caso de prueba para que escriba los registros de la ejecución de la prueba, incluidos los registros del dispositivo que se está probando, en el archivo *`<group-id>`*`_`*`<test-id>`* ubicado en la carpeta *`<device-tester-extract-location>`*`/results/`*`execution-id`*`/logs`. Para ello, recupere la ruta del archivo de registro del contexto de IDT con la consulta `testData.logFilePath`, cree un archivo en esa ruta y escriba el contenido que desee. IDT actualiza automáticamente la ruta en función del caso de prueba que se esté ejecutando. Si decide no crear el archivo de registro para un caso de prueba, no se generará ningún archivo para ese caso de prueba.

También puede configurar el ejecutable de texto para crear archivos de registro adicionales en la carpeta *`<device-tester-extract-location>`*`/logs` según sea necesario. Le recomendamos que especifique prefijos únicos para los nombres de los archivos de registro para que sus archivos no se sobrescriban.

Notificación de los resultados a IDT

IDT escribe los resultados de las pruebas en los archivos `awsiotdevicetester_report.xml` y *`suite-name`*`_report.xml`. Estos archivos de informes están ubicados en *`<device-tester-`*

extract-location/results/*<execution-id>*/. Ambos informes capturan los resultados de la ejecución del conjunto de pruebas. Para obtener más información sobre los esquemas que IDT utiliza para estos informes, consulte Revisión de los resultados y registros de las pruebas de IDT.

Para rellenar el contenido del archivo *suite-name*_report.xml, debe utilizar el comando SendResult para notificar los resultados de las pruebas a IDT antes de que finalice la ejecución de la prueba. Si IDT no puede localizar los resultados de una prueba, emite un error para el caso de prueba. El siguiente extracto de Python muestra los comandos para enviar el resultado de una prueba a IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Si no notifica los resultados a través de la API, IDT busca los resultados de las pruebas en la carpeta de artefactos de la prueba. La ruta a esta carpeta se almacena en la testData.testArtifactsPath indicada en el contexto de IDT. En esta carpeta, IDT utiliza el primer archivo XML ordenado alfabéticamente que localiza como resultado de la prueba.

Si la lógica de la prueba genera resultados XML JUnit, puede escribir los resultados de la prueba en un archivo XML en la carpeta de artefactos para proporcionarlos directamente a IDT, en lugar de analizarlos y, a continuación, utilizar la API para enviarlos a IDT.

Si utiliza este método, asegúrese de que la lógica de la prueba resuma con precisión los resultados de la prueba y formatee el archivo de resultados con el mismo formato que el archivo *suite-name*_report.xml. IDT no realiza ninguna validación de los datos que usted proporciona, con las siguientes excepciones:

- IDT ignora todas las propiedades de la etiqueta testsuites. En su lugar, calcula las propiedades de la etiqueta a partir de los resultados de otros grupos de pruebas notificados.
- Debe haber al menos una etiqueta testsuite en testsuites.

Dado que IDT utiliza la misma carpeta de artefactos para todos los casos de prueba y no elimina los archivos de resultados entre las ejecuciones de las pruebas, este método también puede provocar informes erróneos si IDT lee el archivo incorrecto. Le recomendamos que utilice el mismo nombre para el archivo de resultados XML generado en todos los casos de prueba para sobrescribir los resultados de cada caso de prueba y asegurarse de que IDT pueda utilizar los resultados correctos. Si bien puede utilizar un enfoque mixto para la elaboración de informes en su conjunto de pruebas,

es decir, utilizar un archivo de resultados XML para algunos casos de prueba y enviar los resultados a través de la API para otros, no recomendamos este enfoque.

Especificación del comportamiento de salida

Configure el ejecutable de texto para que siempre salga con un código de salida de 0, incluso si un caso de prueba informa de un fallo o un resultado de error. Utilice códigos de salida distintos de cero únicamente para indicar que un caso de prueba no se ha ejecutado o si el ejecutable del caso de prueba no ha podido comunicar ningún resultado a IDT. Cuando IDT recibe un código de salida distinto de cero, lo marca indicando que el caso de prueba ha detectado un error que ha impedido su ejecución.

IDT podría solicitar o esperar que un caso de prueba deje de ejecutarse antes de que finalice en los siguientes eventos. Utilice esta información para configurar el ejecutable del caso de prueba para que detecte cada uno de estos eventos del caso de prueba:

Timeout (Tiempo de espera

Se produce cuando un caso de prueba se ejecuta durante más tiempo que el valor de tiempo de espera especificado en el archivo `test.json`. Si el ejecutor de la prueba utilizó el argumento `timeout-multiplier` para especificar un multiplicador de tiempo de espera, IDT calcula el valor de tiempo de espera con el multiplicador.

Para detectar este evento, utilice la variable de entorno IDT_TEST_TIMEOUT. Cuando un ejecutor de pruebas lanza una prueba, IDT establece el valor de la variable de entorno IDT_TEST_TIMEOUT en el valor de tiempo de espera calculado (en segundos) y pasa la variable al ejecutable del caso de prueba. Puede leer el valor de la variable para configurar un temporizador adecuado.

Interrumpir

Se produce cuando el ejecutor de pruebas interrumpe IDT. Por ejemplo, pulsando Ctrl+C.

Como los terminales propagan las señales a todos los procesos secundarios, solo tiene que configurar un controlador de señales en sus casos de prueba para detectar las señales de interrupción.

Como alternativa, puede sondear periódicamente la API para comprobar el valor del booleano `CancellationRequested` en la respuesta de la API `PollForNotifications`. Cuando IDT recibe una señal de interrupción, establece el valor del booleano `CancellationRequested` en `true`.

Detención en el primer fallo

Se produce cuando un caso de prueba que se está ejecutando en paralelo con el caso de prueba actual falla y el ejecutor de la prueba utiliza el argumento `stop-on-first-failure` para especificar que IDT debe detenerse cuando encuentra algún error.

Para detectar este evento, puede sondear periódicamente la API para comprobar el valor del booleano `CancellationRequested` en la respuesta de la API `PollForNotifications`. Cuando IDT detecta un fallo y está configurado para detenerse en el primer fallo, establece el valor del booleano `CancellationRequested` en `true`.

Cuando se produce alguno de estos eventos, IDT espera 5 minutos a que los casos de prueba que se están ejecutando terminen de ejecutarse. Si todos los casos de prueba en ejecución no se cierran en 5 minutos, IDT obliga a detener cada uno de sus procesos. Si IDT no ha recibido los resultados de las pruebas antes de que finalicen los procesos, marcará los casos de prueba como tiempo de espera agotado. Como práctica recomendada, debe asegurarse de que los casos de prueba realicen las siguientes acciones cuando detecten alguno de estos eventos:

1. Dejar de ejecutar la lógica de prueba normal.
2. Limpiar todos los recursos temporales, como los artefactos de prueba del dispositivo que se está probando.
3. Notificar el resultado de una prueba a IDT, como un fallo o un error en la prueba.
4. Salir.

Uso del contexto de IDT

Cuando IDT ejecuta un conjunto de pruebas, el conjunto de pruebas puede acceder a un conjunto de datos que se pueden utilizar para determinar cómo se ejecuta cada prueba. Estos datos se denominan contexto de IDT. Por ejemplo, la configuración de los datos de usuario proporcionada por los ejecutores de pruebas en un archivo `userdata.json` se pone a disposición de los conjuntos de pruebas en el contexto de IDT.

El contexto de IDT puede considerarse un documento JSON de solo lectura. Los conjuntos de pruebas pueden recuperar datos del contexto y escribirlos en él mediante tipos de datos JSON estándar, como objetos, matrices, números, etc.

Esquema de contexto

El contexto de IDT utiliza el siguiente formato:

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Información del archivo `config.json`. El campo `config` también contiene el siguiente campo adicional:

`config.timeoutMultiplier`

El multiplicador para cualquier valor de tiempo de espera utilizado por el conjunto de pruebas. El ejecutor de pruebas especifica este valor desde la CLI de IDT. El valor predeterminado es 1.

`device`

Información sobre el dispositivo seleccionado para la prueba. Esta información equivale al elemento de matriz `devices` del archivo `device.json` del dispositivo seleccionado.

`devicePool`

Información sobre el grupo de dispositivos seleccionado para la ejecución de la prueba. Esta información equivale al elemento de matriz del grupo de dispositivos en el nivel superior definido en el archivo `device.json` para el grupo de dispositivos seleccionado.

`resource`

Información sobre los dispositivos de recursos del archivo `resource.json`.

`resource.devices`

Esta información equivale a la matriz `devices` definida en el archivo `resource.json`. Cada elemento `devices` incluye el siguiente campo adicional:

`resource.device.name`

El nombre del dispositivo de recursos. Este valor se establece en el valor `requiredResource.name` en el archivo `test.json`.

`testData.awsCredentials`

Las credenciales de AWS que se utilizan en la prueba para conectarse a la nube de AWS. Esta información se obtiene del archivo `config.json`.

`testData.logFilePath`

La ruta al archivo de registro en el que el caso de prueba escribe los mensajes de registro. El conjunto de pruebas crea este archivo si no existe.

`userData`

Información proporcionada por el ejecutor de la prueba en el archivo `userdata.json`.

Acceda a los datos en el contexto

Puede consultar el contexto mediante la notación JSONPath de sus archivos JSON y de su ejecutable de texto con las API `GetContextValue` y`GetContextString`. La sintaxis de las cadenas JSONPath para acceder al contexto IDT varía de la siguiente manera:

- En `suite.json` y `test.json`, se usa `{{`*`query`*`}}` Es decir, no utilice el elemento raíz `$.` para iniciar la expresión.
- En `test_orchestrator.yaml`, se usa `{{`*`query`*`}}`.

  Si usa la máquina de estados obsoleta, entonces en `state_machine.json`, usa `{{$.`*`query`*`}}`.
- En los comandos de la API, se utiliza *`query`* o `{{$.`*`query`*`}}`, según el comando. Para obtener más información, consulte la documentación en línea en los SDK.

En la siguiente tabla se describen los operadores de una expresión JSONPath típica:

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.childName`	Accesses the child element with name `childName` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `$.config.awsRegion` .
`[start:end]`	Filters elements from an array, retrieving items beginning from the `iniciar` index and going up to the `finales` index, both inclusive.
`[index1, index2, ... , indexN]`	Filters elements from an array, retrieving items from only the specified indices.

Operator	Description
`[?(expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

Para crear expresiones de filtro, utilice la siguiente sintaxis:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

En esta sintaxis:

- `jsonpath` es un JSONPath que utiliza la sintaxis JSON estándar.
- `value` es cualquier valor personalizado que utilice la sintaxis JSON estándar.
- `operator` es uno de los siguientes operadores:
  - < (Menor que)
  - <= (Menor o igual que)
  - == (Igual que)

    Si el JSONPath o el valor de la expresión es un valor de matriz, booleano o de objeto, este es el único operador binario compatible que puede utilizar.
  - >= (Mayor o igual que)
  - > (Mayor que)
  - =~ (Coincidencia de expresión regular). Para usar este operador en una expresión de filtro, el JSONPath o el valor del lado izquierdo de la expresión debe evaluarse como una cadena y el lado derecho debe ser un valor de patrón que siga la sintaxis RE2.

Puede utilizar consultas JSONPath con el formato {{*query*}} como cadenas de marcador de posición dentro de los campos `args` y `environmentVariables` en los archivos `test.json` y dentro de los campos `environmentVariables` en los archivos `suite.json`. IDT realiza una búsqueda contextual y rellena los campos con el valor evaluado de la consulta. Por ejemplo, en el archivo `suite.json`, puede utilizar cadenas de marcadores de posición para especificar los valores de las variables de entorno que cambian con cada caso de prueba e IDT rellenará las variables de entorno con el valor correcto para cada caso de prueba. Sin embargo, cuando se utilizan cadenas de

marcadores de posición en los archivos `test.json` y `suite.json`, las consultas tienen en cuenta las siguientes consideraciones:

- Debe escribir en minúsculas cada vez que aparezca la clave `devicePool` en la consulta. Es decir, utilizar `devicepool` en su lugar.
- Para las matrices, solo puede usar matrices de cadenas. Además, las matrices utilizan un formato `item1, item2,...,itemN` no estándar. Si la matriz contiene solo un elemento, se serializa como `item`, lo que la hace que no se pueda distinguir de un campo de cadena.
- No puede utilizar marcadores de posición para recuperar objetos del contexto.

Debido a estas consideraciones, le recomendamos que, siempre que sea posible, utilice la API para acceder al contexto en su lógica de prueba en lugar de cadenas de marcadores de posición en los archivos `test.json` y `suite.json`. Sin embargo, en algunos casos puede ser más conveniente utilizar marcadores de posición de JSONPath para recuperar cadenas individuales y configurarlas como variables de entorno.

Configuración de los ajustes para los ejecutores de pruebas

Para ejecutar conjuntos de pruebas personalizados, los ejecutores de pruebas deben configurar sus ajustes en función del conjunto de pruebas que desean ejecutar. Los ajustes se especifican en función de las plantillas del archivo de configuración que se encuentran en la carpeta *<device-tester-extract-location>*`/configs/`. Si es necesario, los ejecutores de las pruebas también deben configurar las credenciales de AWS que IDT utilizará para conectarse a la nube de AWS.

Como redactor de pruebas, necesitará configurar estos archivos para depurar su conjunto de pruebas. Debe proporcionar instrucciones a los ejecutores de pruebas para que puedan configurar los siguientes ajustes según sea necesario para ejecutar sus conjuntos de pruebas.

Configurar device.json

El archivo `device.json` contiene información sobre los dispositivos en los que se ejecutan las pruebas (por ejemplo, dirección IP, información de inicio de sesión, sistema operativo y arquitectura de la CPU).

Los ejecutores de pruebas pueden proporcionar esta información mediante el siguiente archivo `device.json` de plantilla que se encuentra en la carpeta *<device-tester-extract-location>*`/configs/`.

```
[
```

```
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
```

```
    }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`sku`

Un valor alfanumérico que identifica de forma única el dispositivo a prueba. El SKU se utiliza para realizar un seguimiento de los dispositivos cualificados.

> **Note**
>
> Si desea enumerar la placa en el catálogo de dispositivos de AWS Partner, el SKU que especifique aquí debe coincidir con el SKU que utilice en el proceso de publicación.

`features`

Opcional. Una matriz que contenga las características compatibles del dispositivo. Las características del dispositivo son valores definidos por el usuario que se configuran en el conjunto de pruebas. Debe proporcionar a los ejecutores de las pruebas información sobre los nombres y valores de las características que desee incluir en el archivo `device.json`. Por ejemplo, si quiere probar un dispositivo que funciona como servidor MQTT para otros dispositivos, puede configurar la lógica de prueba para validar los niveles admitidos específicos para una característica denominada `MQTT_QOS`. Los ejecutores de las pruebas proporcionan el nombre de esta característica y establecen su valor en los niveles de QOS compatibles con su dispositivo. Puede recuperar la información proporcionada desde el contexto de IDT con la consulta `devicePool.features`, o desde el contexto del orquestador de pruebas con la consulta `pool.features`.

`features.name`

El nombre de la característica.

`features.value`

Los valores de la característica admitidos.

`features.configs`

Los ajustes de configuración de la característica, si son necesarios.

`features.config.name`

El nombre del ajuste de configuración.

`features.config.value`

Los valores de configuración admitidos.

`devices`

Una matriz de dispositivos en el grupo que se va a probar. Se requiere al menos un dispositivo.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Cada dispositivo de un grupo debe usar el mismo protocolo.

Actualmente, los únicos valores que se admiten son `ssh` y `uart` para dispositivos físicos y `docker` para contenedores de Docker.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

`connectivity.serialPort`

Opcional. El puerto serie al que está conectado el dispositivo.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `uart`.

`connectivity.containerId`

El ID de contenedor o el nombre del contenedor de Docker que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.containerUser`

Opcional. El nombre del usuario que se va a utilizar dentro del contenedor. El valor predeterminado es el usuario proporcionado en el Dockerfile.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

> **Note**
>
> Para verificar si los ejecutores de pruebas configuran la conexión de dispositivo incorrecta para una prueba, puede recuperar `pool.Devices[0].Connectivity.Protocol` del contexto del orquestador de pruebas y compararlo con el valor esperado en un estado `Choice`. Si se utiliza un protocolo incorrecto, imprima un mensaje con el estado `LogMessage` y haga la transición al estado `Fail`.
> Como alternativa, puede utilizar un código de gestión de errores para informar de un fallo en la prueba para los tipos de dispositivos incorrectos.

(Opcional) Configuración de userdata.json

El archivo `userdata.json` contiene cualquier información adicional que requiera un conjunto de pruebas, pero que no esté especificada en el archivo `device.json`. El formato de este archivo depende del archivo `userdata_scheme.json` definido en el conjunto de pruebas. Si es un redactor de pruebas, asegúrese de proporcionar esta información a los usuarios que van a ejecutar los conjuntos de pruebas que escriba.

(Opcional) Configuración de resource.json

El archivo `resource.json` contiene información sobre los dispositivos que se van a utilizar como dispositivos de recursos. Los dispositivos de recursos son dispositivos que se requieren para probar ciertas capacidades de un dispositivo que se está probando. Por ejemplo, para probar la capacidad Bluetooth de un dispositivo, puede usar un dispositivo de recursos para comprobar si el dispositivo se puede conectar correctamente a él. Los dispositivos de recursos son opcionales y puede requerir tantos dispositivos de recursos como necesite. Como redactor de pruebas, utilice el archivo test.json para definir las características del dispositivo de recursos que se requieren para una prueba. A

continuación, los ejecutores de pruebas utilizan el archivo `resource.json` para proporcionar un grupo de dispositivos de recursos que tengan las características necesarias. Asegúrese de proporcionar esta información a los usuarios que vayan a ejecutar los conjuntos de pruebas que escriba.

Los ejecutores de pruebas pueden proporcionar esta información mediante el siguiente archivo `resource.json` de plantilla que se encuentra en la carpeta *<device-tester-extract-location>*`/configs/`.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-version>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
```

```
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

`id`

Un ID alfanumérico definido por el usuario que identifica de forma única una colección de dispositivos que se conoce como grupo de dispositivos. Los dispositivos que pertenecen a un grupo deben tener idéntico hardware. Al ejecutar un conjunto de pruebas, los dispositivos del grupo se utilizan para paralelizar la carga de trabajo. Se utilizan varios dispositivos para ejecutar diferentes pruebas.

`features`

Opcional. Una matriz que contenga las características compatibles del dispositivo. La información requerida en este campo se define en los archivos test.json del conjunto de pruebas y determina qué pruebas se van a ejecutar y cómo se van a ejecutar. Si el conjunto de pruebas no requiere ninguna característica, este campo no es obligatorio.

`features.name`

El nombre de la característica.

`features.version`

La versión de la característica.

`features.jobSlots`

Configuración para indicar cuántas pruebas pueden utilizar el dispositivo simultáneamente. El valor predeterminado es 1.

`devices`

Una matriz de dispositivos en el grupo que se va a probar. Se requiere al menos un dispositivo.

`devices.id`

Un identificador único y definido por el usuario para el dispositivo que se está probando.

`connectivity.protocol`

El protocolo de comunicación que se usará para la comunicación con este dispositivo. Cada dispositivo de un grupo debe usar el mismo protocolo.

Actualmente, los únicos valores que se admiten son `ssh` y `uart` para dispositivos físicos y `docker` para contenedores de Docker.

`connectivity.ip`

La dirección IP del dispositivo que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.port`

Opcional. El número de puerto que se va a utilizar para las conexiones SSH.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth`

Información de autenticación para la conexión.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.auth.method`

El método de autenticación que se utiliza para acceder a un dispositivo a través de un determinado protocolo de conectividad.

Los valores admitidos son:

- `pki`
- `password`

`connectivity.auth.credentials`

Las credenciales que se utilizan para la autenticación.

`connectivity.auth.credentials.password`

La contraseña que se utiliza para iniciar sesión en el dispositivo que se va a probar.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `password`.

`connectivity.auth.credentials.privKeyPath`

La ruta completa a la clave privada que se utiliza para iniciar sesión en el dispositivo que se está probando.

Este valor solo se aplica si `connectivity.auth.method` está establecido en `pki`.

`connectivity.auth.credentials.user`

El nombre de usuario para iniciar sesión en el dispositivo que se está probando.

`connectivity.serialPort`

Opcional. El puerto serie al que está conectado el dispositivo.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `uart`.

`connectivity.containerId`

El ID de contenedor o el nombre del contenedor de Docker que se está probando.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

`connectivity.containerUser`

Opcional. El nombre del usuario que se va a utilizar dentro del contenedor. El valor predeterminado es el usuario proporcionado en el Dockerfile.

El valor predeterminado es 22.

Esta propiedad solo se aplica si `connectivity.protocol` está establecido en `ssh`.

(Opcional) Configuración de config.json

El archivo `config.json` contiene la información de configuración para IDT. Por lo general, los ejecutores de pruebas no necesitarán modificar este archivo excepto para proporcionar sus credenciales de usuario de AWS para IDT y, opcionalmente, una región de AWS. Si se proporcionan las credenciales de AWS con los permisos necesarios, AWS IoT Device Tester recopila y envía las métricas de uso a AWS. Se trata de una característica opcional que se utiliza para mejorar la funcionalidad de IDT. Para obtener más información, consulte Métricas de uso de IDT.

Los ejecutores de pruebas pueden configurar sus credenciales de AWS de una de las siguientes maneras:

- Archivo de credenciales

  IDT utiliza el mismo archivo de credenciales que la AWS CLI. Para obtener más información, consulte Configuración y archivos de credenciales.

  La ubicación del archivo de credenciales varía en función del sistema operativo que utilice:

  - macOS, Linux: `~/.aws/credentials`
  - Windows: : `C:\Users\UserName\.aws\credentials`
- Variables de entorno

  Las variables de entorno son las variables que mantiene el sistema operativo y utilizan los comandos del sistema. Las variables definidas durante una sesión de SSH no están disponibles una vez cerrada la sesión. IDT puede usar las variables de entorno `AWS_ACCESS_KEY_ID` y `AWS_SECRET_ACCESS_KEY` para almacenar sus credenciales de AWS.

  Para establecer estas variables en Linux, MacOS, o Unix, utilice : export:

  ```
  export AWS_ACCESS_KEY_ID=<your_access_key_id>
  export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
  ```

  Para establecer estas variables en Windows, utilice set:

  ```
  set AWS_ACCESS_KEY_ID=<your_access_key_id>
  set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
  ```

Para configurar las credenciales de AWS para IDT, los ejecutores de pruebas editan la sección `auth` del archivo `config.json` ubicado en la carpeta *<device-tester-extract-location>*`/configs/`.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
```

```
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Todos los campos que contienen valores son obligatorios tal y como se describe aquí:

> **Note**
>
> Todas las rutas de este archivo se definen en relación con *<device-tester-extract-location>*.

`log.location`

La ruta a la carpeta de registros en *<device-tester-extract-location>*.

`configFiles.root`

La ruta a la carpeta que contiene los archivos de configuración.

`configFiles.device`

La ruta al archivo `device.json`.

`testPath`

La ruta a la carpeta que contiene los conjuntos de pruebas.

`reportPath`

La ruta a la carpeta que contendrá los resultados de las pruebas después de que IDT ejecute un conjunto de pruebas.

`awsRegion`

Opcional. La región de AWS que utilizarán los conjuntos de pruebas. Si no se establece, los conjuntos de pruebas utilizarán la región predeterminada especificada en cada conjunto de pruebas.

`auth.method`

El método que IDT utiliza para recuperar las credenciales de AWS. Los valores admitidos son `file` para recuperar las credenciales de un archivo de credenciales y `environment` para recuperar las credenciales mediante variables de entorno.

`auth.credentials.profile`

El perfil de credenciales que se va a utilizar del archivo de credenciales. Esta propiedad solo se aplica si `auth.method` está establecido en `file`.

Depurar y ejecutar conjuntos de pruebas personalizadas

Una vez establecida la configuración requerida, IDT puede ejecutar su conjunto de pruebas. El tiempo de ejecución del conjunto de pruebas completa depende del hardware y de la composición del conjunto de pruebas. Como referencia, se tarda aproximadamente 30 minutos en completar el conjunto de pruebas de AWS IoT Greengrass completo en una unidad Raspberry Pi 3B.

Mientras escribe su conjunto de pruebas, puede usar IDT para ejecutarla en modo de depuración, comprobar el código antes de ejecutarla o proporcionárselo a los ejecutores de pruebas.

Ejecución de IDT en modo de depuración

Como los conjuntos de pruebas dependen de IDT para interactuar con los dispositivos, proporcionar el contexto y recibir los resultados, no puede simplemente depurar sus conjuntos de pruebas en un IDE sin ninguna interacción con IDT. Para ello, la CLI de IDT proporciona el comando `debug-test-suite`, que permite ejecutar IDT en modo de depuración. Ejecute el siguiente comando para ver las opciones disponibles para `debug-test-suite`:

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Cuando se ejecuta IDT en modo de depuración, IDT no inicia realmente el conjunto de pruebas ni ejecuta orquestador de pruebas, sino que interactúa con el IDE para responder a las solicitudes realizadas desde el conjunto de pruebas que se ejecuta en el IDE e imprime los registros en la consola. IDT no agota el tiempo de espera y espera a salir hasta que se interrumpa manualmente. En el modo de depuración, IDT tampoco ejecuta el orquestador de pruebas y no generará ningún archivo de informe. Para depurar su conjunto de pruebas, debe usar su IDE para proporcionar cierta información que IDT suele obtener de los archivos JSON de configuración. Asegúrese de que proporciona la siguiente información:

- Variables de entorno y argumentos para cada prueba. IDT no leerá esta información de `test.json` ni `suite.json`.
- Argumentos para seleccionar los dispositivos de recursos. IDT no leerá esta información de `test.json`.

Para depurar los conjuntos de pruebas, complete los pasos siguientes:

1. Cree los archivos de configuración de ajustes necesarios para ejecutar el conjunto de pruebas. Por ejemplo, si su conjunto de pruebas requiere `device.json`, `resource.json`, y `user data.json`, asegúrese de configurarlos todos según sea necesario.
2. Ejecute el siguiente comando para establecer IDT en modo de depuración y seleccione los dispositivos necesarios para ejecutar la prueba.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Tras ejecutar este comando, IDT espera las solicitudes del conjunto de pruebas y, a continuación, responde a ellas. IDT también genera las variables de entorno que se requieran para el proceso de casos para el SDK de cliente de IDT.
3. En su IDE, utilice la configuración `run` o `debug` para hacer lo siguiente:
   a. Establecer los valores de las variables de entorno generadas por IDT.
   b. Establecer el valor de cualquier variable de entorno o argumento que haya especificado en el archivo `test.json` y `suite.json`.
   c. Establecer los puntos de interrupción según sea necesario.
4. Ejecute el conjunto de pruebas en su IDE.

   Puede depurar y volver a ejecutar el conjunto de pruebas tantas veces como sea necesario. En el modo de depuración, IDT no agota el tiempo de espera.
5. Una vez completada la depuración, interrumpa IDT para salir del modo de depuración.

Comandos de la CLI de IDT para ejecutar pruebas

En la sección siguiente se describen los comandos de la CLI de IDT.

IDT v4.0.0

`help`

Enumera información acerca del comando especificado.

`list-groups`

Muestra los grupos de un conjunto de prueba determinado.

`list-suites`

Muestra los conjuntos de prueba disponibles.

`list-supported-products`

Enumera los productos compatibles con su versión de IDT, en este caso las versiones AWS IoT Greengrass de cualificaciones AWS IoT Greengrass y las versiones del conjunto de pruebas para la versión actual de IDT.

`list-test-cases`

Enumera los casos de prueba en un grupo de prueba determinado. Se admite la siguiente opción:

- `group-id`. El grupo de pruebas que se va a buscar. Esta opción es necesaria y debe especificar un solo grupo.

`run-suite`

Ejecuta un conjunto de pruebas en un grupo de dispositivos. Estas son algunas opciones que suelen utilizarse:

- `suite-id`. La versión del conjunto de pruebas que se va a ejecutar. Si no se especifica, IDT utiliza la versión más reciente de la carpeta `tests`.
- `group-id`. Los grupos de pruebas que se van a ejecutar, como una lista separada por comas. Si no se especifica, IDT ejecuta todos los grupos de prueba del conjunto de pruebas.
- `test-id`. Los casos de prueba que se van a ejecutar, como una lista separada por comas. Cuando se especifique, `group-id` debe especificar un solo grupo.
- `pool-id`. El grupo de dispositivos que se va a probar. Los ejecutores de las pruebas deben especificar un grupo si tienen varios grupos de dispositivos definidos en el archivo `device.json`.

- `timeout-multiplier`. Configura IDT para modificar el tiempo de espera de ejecución de la prueba especificado en el archivo `test.json` para una prueba con un multiplicador definido por el usuario.
- `stop-on-first-failure`. Configura IDT para detener la ejecución en el primer error. Esta opción debe utilizarse con `group-id` para depurar los grupos de prueba especificados.
- `userdata`. Establece el archivo que contiene la información sobre los datos del usuario necesarios para ejecutar el conjunto de pruebas. Esto solo es necesario si `userdataRequired` está establecido en verdadero en el archivo `suite.json` del conjunto de pruebas.

Para obtener más información acerca de `run-suite` las opciones, utilice la opción `help`:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

Ejecute el conjunto de pruebas en modo de depuración. Para obtener más información, consulte Ejecución de IDT en modo de depuración.

Revisión de los resultados y registros de las pruebas de IDT

En esta sección se describe el formato en que IDT genera los registros de la consola y los informes de las pruebas.

Formato de mensajes de consola

AWS IoT Device Tester utiliza un formato estándar para imprimir mensajes en la consola cuando inicia un conjunto de pruebas. En el fragmento siguiente se muestra un ejemplo de mensaje de consola generado por IDT.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La mayoría de los mensajes de consola constan de los siguientes campos:

`time`

Una marca de tiempo completa conforme a la norma ISO 8601 para el evento registrado.

`level`

El nivel de mensaje del evento registrado. Normalmente, el nivel del mensaje registrado es uno de los siguientes: `info`, `warn` o `error`. IDT emite un mensaje `panic` o `fatal` si detecta un evento esperado que provoca su cierre anticipado.

`msg`

El mensaje registrado.

`executionId`

Una cadena de ID único para el proceso de IDT actual. Este ID se utiliza para diferenciar entre ejecuciones de IDT individuales.

Los mensajes de consola generados a partir de un conjunto de pruebas proporcionan información adicional sobre el dispositivo que se está probando y el conjunto de pruebas, el grupo de pruebas y los casos de prueba que ejecuta IDT. En el fragmento siguiente se muestra un ejemplo de un mensaje de consola generado por un conjunto de pruebas.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La parte específica del mensaje de la consola para el conjunto de pruebas contiene los siguientes campos:

`suiteId`

El nombre del conjunto de pruebas que se está ejecutando.

`groupId`

El ID del grupo de pruebas que se está ejecutando.

`testCaseId`

El ID del caso de prueba que se está ejecutando.

`deviceId`

Un ID del dispositivo que se está probando y que el caso de prueba está utilizando.

Para imprimir un resumen de la prueba en la consola cuando IDT termina de ejecutar una prueba, debe incluir un estado de `Report` en el orquestador de pruebas. El resumen de la prueba contiene información sobre el conjunto de pruebas, los resultados de las pruebas de cada grupo que se ejecutó y las ubicaciones de los registros y archivos de informes generados. En el siguiente ejemplo se muestra un mensaje de resumen de la prueba.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
---------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
---------------------------------
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

Esquema de informe de AWS IoT Device Tester

`awsiotdevicetester_report.xml` es un informe firmado que contiene la siguiente información:

- La versión de IDT.
- La versión del conjunto de pruebas.
- La firma del informe y la clave utilizada para firmarlo.
- El SKU del dispositivo y el grupo de dispositivos especificado en el archivo `device.json`.
- La versión del producto y las características del dispositivo que se han probado.
- El resumen de agregación de los resultados de las pruebas. Esta información es la misma que la que se incluye en el archivo *`suite-name`*`_report.xml`.

```
<apnreport>
```

```
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

El archivo `awsiotdevicetester_report.xml` contiene una etiqueta `<awsproduct>` que tiene información sobre el producto que se está probando y las características del producto que se han validado después de ejecutar un conjunto de pruebas.

Atributos que se utilizan en la etiqueta **`<awsproduct>`**

`name`

El nombre del producto que se está probando.

`version`

La versión del producto que se está probando.

`features`

Las características validadas. Las características marcadas como `required` son necesarias para que el conjunto de pruebas valide el dispositivo. En el siguiente fragmento se muestra cómo aparece esta información en el archivo `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Las funciones marcadas como `optional` no son necesarias para la validación. Los siguientes fragmentos muestran características opcionales:

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Esquema del informe del conjunto de pruebas

El informe *`suite-name`*`_Result.xml` está en formato XML JUnit. Puede integrarlo en plataformas de integración/implementación continua como Jenkins, Bamboo, etc. El informe contiene un resumen global de los resultados de las pruebas.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
```

```
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

La sección de informe tanto en `awsiotdevicetester_report.xml` como en *`suite-name`*`_report.xml` enumera las pruebas que se han ejecutado y los resultados.

La primera etiqueta XML `<testsuites>` contiene el resumen de la ejecución de las pruebas. Por ejemplo:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atributos que se utilizan en la etiqueta **`<testsuites>`**

`name`

El nombre del grupo de prueba.

`time`

El tiempo, en segundos, que se ha tardado en ejecutar el conjunto de pruebas.

`tests`

El número de pruebas ejecutadas.

`failures`

El número de pruebas que se ejecutaron, pero que no se superaron.

`errors`

El número de pruebas que IDT no ha podido ejecutar.

`disabled`

Este atributo no se utiliza y se puede omitir.

Si se producen errores en pruebas, puede identificar la prueba fallido revisando las etiquetas XML `<testsuites>`. Las etiquetas XML `<testsuite>` dentro de la etiqueta `<testsuites>` muestran el resumen del resultado de la prueba de un grupo de prueba. Por ejemplo:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

El formato es similar a la etiqueta `<testsuites>`, pero con un atributo `skipped` que no se utiliza y que se puede pasar por alto. Dentro de cada etiqueta XML `<testsuite>`, hay etiquetas `<testcase>` para cada prueba ejecutada para un grupo de prueba. Por ejemplo:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Atributos que se utilizan en la etiqueta **`<testcase>`**

`name`

El nombre de la prueba.

`attempts`

El número de veces que IDT ha ejecutado el caso de prueba.

Cuando una prueba genera un error o si se produce un error, las etiquetas `<failure>` o `<error>` se agregan a la etiqueta `<testcase>` con información para la resolución de problemas. Por ejemplo:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Métricas de uso de IDT

Si proporciona AWS credenciales con los permisos necesarios, AWS IoT Device Tester recopila y envía las métricas de uso a AWS. Se trata de una característica opcional que se utiliza para mejorar la funcionalidad de IDT. IDT recopila información como la siguiente:

- El Cuenta de AWS ID utilizado para ejecutar IDT
- Los AWS CLI comandos de IDT utilizados para ejecutar las pruebas
- El conjunto de pruebas que se ejecutan
- Los conjuntos de pruebas de la carpeta *<device-tester-extract-location>*
- La cantidad de dispositivos configurados en el grupo de dispositivos
- Los nombres de casos de prueba y los tiempos de ejecución
- La información sobre los resultados de las pruebas, por ejemplo, si se han superado, si han fallado, si se han encontrado errores o si se han omitido
- Las características del producto probadas
- El comportamiento de salida de IDT, como salidas inesperadas o anticipadas

Toda la información que IDT envía también se registra en un archivo `metrics.log` de la carpeta *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Puede consultar el archivo de registro para ver la información recopilada durante la ejecución de una prueba. Este archivo se genera solo si elige recopilar métricas de uso.

Para deshabilitar la recopilación de métricas, no es necesario que realice ninguna acción adicional. Simplemente no almacene sus AWS credenciales y, si AWS las tiene almacenadas, no configure el `config.json` archivo para acceder a ellas.

Configure sus AWS credenciales

Si aún no tiene una Cuenta de AWS, debe crear una. Si ya tiene una Cuenta de AWS, solo tiene que configurar los permisos necesarios para su cuenta para que IDT pueda enviarle las métricas de uso AWS en su nombre.

Paso 1: Crea una Cuenta de AWS

En este paso, cree y configure una Cuenta de AWS. Si ya tiene una Cuenta de AWS, vaya directamente a the section called “Paso 2: Configurar los permisos de IDT”.

Si no tiene uno Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abrir https://portal.aws.amazon.com/billing/registro.
2. Siga las instrucciones que se le indiquen.

Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto e indicar un código de verificación en el teclado del teléfono.

Cuando te registras en un Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario y utilice únicamente el usuario raíz para realizar Tareas que requieren acceso de usuario raíz.

Para crear un usuario administrador, elija una de las siguientes opciones.

Elegir una forma de administrar el administrador	Para	Haga esto	También puede
En IAM Identity Center (recomendado)	Usar credenciales a corto plazo para acceder a AWS. Esto se ajusta a las prácticas recomendadas de seguridad. Para obtener información sobre las prácticas recomendadas, consulta Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.	Siga las instrucciones en Introducción en la Guía del usuario de AWS IAM Identity Center .	Configure el acceso programático configurando el AWS CLI que se utilizará AWS IAM Identity Center en la Guía del AWS Command Line Interface usuario.

Elegir una forma de administrar el administrador	Para	Haga esto	También puede
En IAM (no recomendado)	Usar credenciales a largo plazo para acceder a AWS.	Siguiendo las instrucciones de Crear un usuario de IAM para acceso de emergencia de la Guía del usuario de IAM.	Configure el acceso programático mediante Administrar las claves de acceso de los usuarios de IAM en la Guía del usuario de IAM.

Paso 2: Configurar los permisos de IDT

En este paso, configure los permisos que IDT utiliza para ejecutar las pruebas y recopilar datos de uso de IDT. Puede usar Consola de administración de AWS o AWS Command Line Interface (AWS CLI) para crear una política de IAM y un usuario para IDT y, a continuación, adjuntar políticas al usuario.

- Configuración de permisos para IDT (consola)
- Configuración de permisos para IDT (AWS CLI)

Configuración de permisos de IDT (consola)

Siga estos pasos para usar la consola para configurar permisos para IDT para AWS IoT Greengrass.

1. Inicie sesión en la consola de IAM.
2. Crear una política administrada que conceda permisos para crear roles con permisos específicos.

   a. En el panel de navegación, seleccione Políticas y, a continuación, Crear política.

   b. En la pestaña JSON, reemplace el contenido del marcador de posición por la política siguiente.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

c. Elija Revisar política.

d. En Nombre, ingrese **`IDTUsageMetricsIAMPermissions`**. En Summary (Resumen), revise los permisos concedidos por la política.

e. Elija Crear política.

3. Cree un usuario de IAM y adjunte los permisos al usuario.

a. Cree un usuario de IAM. Siga los pasos del 1 al 5 en Creación de usuarios de IAM (consola) en la Guía del usuario de IAM. Si ya ha creado un usuario de IAM, pase directamente al siguiente paso.

b. Adjunte los permisos a su usuario de IAM:

i. En la página Set permissions (Establecer permisos), elija Attach existing policies to user directly (Adjuntar políticas existentes al usuario directamente).

ii. Busque la IAMPermissions política de IDTUsagemétricas que creó en el paso anterior. Seleccione la casilla de verificación.

c. Elija Siguiente: etiquetas.

d. Elija Next: Review (Siguiente: revisar) para ver un resumen de sus opciones.

e. Seleccione la opción Crear un usuario.

f. Para ver las claves de acceso del usuario (clave de acceso IDs y claves de acceso secretas), selecciona Mostrar junto a la contraseña y la clave de acceso. Para guardar las claves de acceso, elija Download.csv (Descargar archivo .csv) y, a continuación, guarde

el archivo en un lugar seguro. Utilizará esta información más adelante para configurar el archivo de AWS credenciales.

Configuración de permisos de IDT (AWS CLI)

Siga estos pasos AWS CLI para configurar los permisos de IDT para AWS IoT Greengrass.

1. En su ordenador, instale y configure el AWS CLI si aún no está instalado. Siga los pasos que se indican en Instalación de la AWS CLI en la Guía del usuario de la AWS Command Line Interface .

   > **Note**
   >
   > AWS CLI Se trata de una herramienta de código abierto que puede utilizar para interactuar con los AWS servicios desde el shell de la línea de comandos.

2. Cree la siguiente política gestionada por el cliente que conceda permisos para gestionar el IDT y las funciones. AWS IoT Greengrass

   Linux or Unix

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "iot-device-tester:SendMetrics"
               ],
               "Resource": "*"
           }
       ]
   }'
   ```

   Windows command prompt

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document
   ```

```
                                        '{\"Version\": \"2012-10-17\",
     \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
tester:SendMetrics\"], \"Resource\": \"*\"}]}'
```

> **Note**
>
> Este paso incluye un ejemplo de símbolo del sistema de Windows porque utiliza una sintaxis JSON diferente a la de los comandos de terminal Linux, macOS o Unix.

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

3. Cree un usuario de IAM y adjunte los permisos requeridos por IDT para AWS IoT Greengrass.

   a. Cree un usuario de IAM.

   ```
   aws iam create-user --user-name user-name
   ```

   b. Adjunte la política `IDTUsageMetricsIAMPermissions` que ha creado a su nuevo usuario de IAM. *user-name*Sustitúyalo por tu nombre de usuario de IAM y, *<account-id>* en el comando, por tu ID. Cuenta de AWS

   ```
   aws iam attach-user-policy --user-name user-name --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

4. Cree una clave de acceso secreta para el usuario.

```
aws iam create-access-key --user-name user-name
```

Almacene la salida en una ubicación segura. Utilizará esta información más adelante para configurar el archivo de AWS credenciales.

Proporcione AWS las credenciales a IDT

Para permitir que IDT acceda a sus AWS credenciales y envíe las métricas a ellas AWS, haga lo siguiente:

1. Guarde las AWS credenciales de su usuario de IAM como variables de entorno o en un archivo de credenciales:

   a. Para usar variables de entorno, ejecute los siguientes comandos.

      Linux or Unix

      ```
      export AWS_ACCESS_KEY_ID=access-key
      export AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      Windows Command Prompt (CMD)

      ```
      set AWS_ACCESS_KEY_ID=access-key
      set AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      PowerShell

      ```
      $env:AWS_ACCESS_KEY_ID="access-key"
      $env:AWS_SECRET_ACCESS_KEY="secret-access-key"
      ```

   b. Para utilizar el archivo de credenciales, agregue la siguiente información al archivo `~/.aws/credentials`.

      ```
      [profile-name]
      aws_access_key_id=access-key
      aws_secret_access_key=secret-access-key
      ```

2. Configure la sección `auth` del archivo `config.json`. Para obtener más información, consulte (Opcional) Configuración de config.json.

Solución de problemas de IDT para V2 AWS IoT Greengrass

IDT para AWS IoT Greengrass V2 escribe los errores en varias ubicaciones según el tipo de error. IDT escribe errores en la consola, en archivos de registro y en informes de prueba.

¿Dónde puedo buscar los errores?

Los errores generales se muestran en la consola durante la ejecución y se muestra un resumen de las pruebas fallidas con el error una vez completadas todas las pruebas. `awsiotdevicetester_report.xml` contiene un resumen de todos los errores que han provocado fallos en una prueba. IDT almacena los archivos de registro de cada ejecución de prueba en un directorio con un UUID para la ejecución de la prueba, que se muestra en la consola durante la ejecución.

El directorio de registros de pruebas de IDT es *<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`. Este directorio contiene los siguientes archivos que se muestran en la tabla. Esto es útil a efectos de depuración.

Archivos	Description (Descripción)
`test_manager.log`	Los registros escritos en la consola mientras se estaba ejecutando la prueba. El resumen de los resultados al final de este archivo incluye una lista de las pruebas fallidas. La advertencia y los registros de errores en este archivo pueden proporcionarle información acerca de los errores que se producen.
test-group-id /*test-case-id* /*test-name* `.log`	Registros detallados de la prueba específica en un grupo de prueba. En el caso de las pruebas que implementan componentes de Greengras s, el archivo de registro de casos de prueba se denomina `greengrass-test-run.log` .
test-group-id /*test-case-id* / `greengrass.log`	Registros detallados del software AWS IoT Greengrass Core. IDT copia este archivo del dispositivo que se está probando cuando

Archivos	Description (Descripción)
	ejecuta pruebas en las que se instala el software AWS IoT Greengrass Core en el dispositivo. Para obtener más información sobre los mensajes de este archivo de registro, consulte Solución de problemas AWS IoT Greengrass V2.
test-group-id /*test-case-id*/*component-name* .log	Registros detallados de los componentes de Greengrass que se implementan durante las pruebas. IDT copia los archivos de registro de los componentes del dispositivo que se está probando cuando ejecuta pruebas en las que se implementan componentes específicos. El nombre de cada archivo de registro de componentes corresponde al nombre del componente implementado. Para obtener más información sobre los mensajes de este archivo de registro, consulte Solución de problemas AWS IoT Greengrass V2.

Resolución de errores de IDT para la V2 AWS IoT Greengrass

Antes de ejecutar IDT for AWS IoT Greengrass, coloque los archivos de configuración correctos. Si recibe errores de análisis y configuración, lo primero que debe hacer es buscar y utilizar una plantilla de configuración adecuada para su entorno.

Si continúa teniendo problemas, consulte el siguiente proceso de depuración.

Temas

- Errores de resolución de alias
- Errores de conflicto
- Error por la imposibilidad de iniciar una prueba
- La imagen de calificación de Docker contiene errores
- No se pudo leer la credencial

- Errores de guía con Greengrass PreInstalled
- Excepción de firma no válida
- Errores de calificación de machine learning
- Implementaciones fallidas de Open Test Framework (OTF)
- Errores de procesamiento
- Errores de permiso denegado
- Error al generar el informe de calificación
- Error por ausencia de un parámetro obligatorio
- Excepción de seguridad en macOS
- Errores de conexión SSH
- Errores de calificación del administrador de flujos
- Errores de tiempo de espera
- Errores de comprobación de versiones

Errores de resolución de alias

Al ejecutar conjuntos de pruebas personalizados, es posible que aparezca el siguiente error en la consola y en el `test_manager.log`.

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

Este error puede producirse cuando los alias configurados en el orquestador de pruebas de IDT no se resuelven correctamente o si los valores resueltos no están presentes en los archivos de configuración. Para resolver este error, asegúrese de que `device.json` y `userdata.json` contenga la información correcta requerida para su conjunto de pruebas. Para obtener información sobre la configuración necesaria para la AWS IoT Greengrass cualificación, consulteConfigure los ajustes de IDT para ejecutar el conjunto de AWS IoT Greengrass cualificación.

Errores de conflicto

Es posible que aparezca el siguiente error al ejecutar el conjunto de AWS IoT Greengrass requisitos de forma simultánea en más de un dispositivo.

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-
id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409,
```

```
RequestID: “id” }, Message_: “Component [com.example.IDTHelloWorld : 1.0.0] for
account [account-id] already exists with state: [DEPLOYABLE]” }
```

El conjunto de AWS IoT Greengrass calificaciones aún no admite la ejecución simultánea de pruebas. Ejecute el conjunto de calificación de forma secuencial para cada dispositivo.

Error por la imposibilidad de iniciar una prueba

Es posible que encuentre errores que apunten a fallos que se produjeron cuando la prueba intentaba comenzar. Existen varias causas posibles, por lo que debe hacer lo siguiente:

- Asegúrese de que el nombre del grupo en el comando de ejecución existe realmente. IDT hace referencia al nombre del grupo directamente desde el archivo `device.json`.
- Asegúrese de que el dispositivo o dispositivos del grupo tienen parámetros de configuración correctos.

La imagen de calificación de Docker contiene errores

Las pruebas de calificación del administrador de aplicaciones de Docker utilizan la imagen del contenedor `amazon/amazon-ec2-metadata-mock` en Amazon ECR para calificar el dispositivo sometido a prueba.

Es posible que reciba el siguiente error si la imagen ya está presente en un contenedor de Docker del dispositivo que se está probando.

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

Si anteriormente descargó esta imagen y ejecutó el contenedor `amazon/amazon-ec2-metadata-mock` en su dispositivo, asegúrese de eliminar esta imagen del dispositivo objeto de la prueba antes de realizar las pruebas de calificación.

No se pudo leer la credencial

Al probar dispositivos Windows, es posible que aparezca el error `Failed to read credential` en el archivo `greengrass.log` si el usuario que utiliza para conectarse al dispositivo que se está probando no está configurado en el administrador de credenciales de ese dispositivo.

Para resolver este error, configure el usuario y la contraseña del usuario de IDT en el administrador de credenciales del dispositivo que se está probando.

Para obtener más información, consulte Configuración de las credenciales de usuario para los dispositivos Windows.

Errores de guía con Greengrass PreInstalled

Al ejecutar IDT con PreInstalled Greengrass, si encuentra un error `Guice ErrorInCustomProvider` o compruebe si el `userdata.json` archivo está configurado en la carpeta de `InstalledDirRootOnDevice` instalación de Greengrass. IDT busca el archivo `effectiveConfig.yaml` que aparece en `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`.

Para obtener más información, consulte Configuración de las credenciales de usuario para los dispositivos Windows.

Excepción de firma no válida

Al ejecutar las pruebas de calificación de Lambda, es posible que se produzca el error `invalidsignatureexception` si la máquina host de IDT tiene problemas de acceso a la red. Reinicie el enrutador y vuelva a ejecutar las pruebas.

Errores de calificación de machine learning

Al ejecutar pruebas de calificación de aprendizaje automático (ML), es posible que se produzcan errores de calificación si el dispositivo no cumple con los requisitos para implementar los AWS componentes de aprendizaje automático proporcionados. Para solucionar los errores de calificación de machine learning, haga lo siguiente:

- Busque los detalles de los errores en los registros de los componentes que se implementaron durante la ejecución de la prueba. Los registros del componente se encuentran en el directorio *<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`*<test-group-id>*.
- Agregue el argumento `-Dgg.persist=installed.software` al archivo `test.json` para el caso de prueba fallido. El archivo `test.json` se encuentra en *<device-tester-extract-location>*`/tests/GGV2Q_`*version* `directory. .`

Implementaciones fallidas de Open Test Framework (OTF)

Si las pruebas de OTF no completan la implementación, una causa probable pueden ser los permisos establecidos para la carpeta principal de `TempResourcesDirOnDevice` y

`InstallationDirRootOnDevice`. Para configurar correctamente los permisos de esta carpeta, ejecute el siguiente comando. Sustituya *`folder-name`* por el nombre de la carpeta principal.

```
sudo chmod 755 folder-name
```

Errores de procesamiento

Los errores tipográficos en una configuración de JSON pueden provocar errores de análisis. En la mayoría de los casos, el problema es resultado de omitir un paréntesis, una coma o unas comillas en el archivo JSON. IDT realiza la validación JSON e imprime información de depuración. Imprime la línea en la que se produjo el error, el número de línea y el número de la columna del error de sintaxis. Esta información debería ser suficiente para ayudarte a corregir el error, pero si sigues sin poder localizarlo, puedes realizar la validación manualmente en tu IDE, en un editor de texto como Atom o Sublime, o mediante una herramienta en línea similar JSONLint.

Errores de permiso denegado

IDT realiza operaciones en diversos directorios y archivos en un dispositivo que se está probando. Algunas de estas operaciones requieren acceso raíz. Para automatizar estas operaciones, IDT debe ser capaz de ejecutar comandos con sudo sin escribir una contraseña.

Siga estos pasos para permitir acceso sudo sin escribir una contraseña.

> **Note**
>
> `user` y `username` hacen referencia al usuario SSH que utiliza IDT para acceder al dispositivo a prueba.

1. Use sudo usermod -aG sudo ***`<ssh-username>`*** para añadir el usuario SSH al grupo sudo.
2. Cierre la sesión y, a continuación, vuelva a iniciar sesión para que los cambios surtan efecto.
3. Añada el archivo `/etc/sudoers` y, a continuación, agregue la siguiente línea al final del archivo: *`<ssh-username>`* `ALL=(ALL) NOPASSWD: ALL`

   > **Note**
   >
   > Le recomendamos que utilice sudo visudo al editar `/etc/sudoers`.

Error al generar el informe de calificación

IDT es compatible con las cuatro *major*.*minor* versiones más recientes del paquete de calificación AWS IoT Greengrass V2 (GGV2Q) para generar informes de calificación que puede enviar AWS Partner Network para incluir sus dispositivos en el catálogo de AWS Partner dispositivos. Las versiones anteriores del paquete de calificaciones no generaban informes de calificación.

Si tiene alguna pregunta acerca de la política de compatibilidad, póngase en contacto con AWS Support.

Error por ausencia de un parámetro obligatorio

Cuando IDT agrega nuevas características, puede introducir cambios en los archivos de configuración. Utilizar un archivo de configuración antiguo podría romper la configuración. Si esto ocurre, el archivo *<test_case_id>*.log en /results/*<execution-id>*/logs enumera explícitamente todos los parámetros que faltan. IDT también valida los esquemas del archivo de configuración JSON para asegurarse de que se ha utilizado la última versión compatible.

Excepción de seguridad en macOS

Cuando ejecuta IDT en una computadora host macOS, se bloquea la ejecución de IDT. Para ejecutar IDT, conceda una excepción de seguridad a los ejecutables que forman parte de la funcionalidad de tiempo de ejecución de IDT. Cuando aparezca el mensaje de advertencia en el equipo host, haga lo siguiente para cada uno de los ejecutables aplicables:

Cómo conceder una excepción de seguridad de IDT ejecutable

1. En la computadora macOS, en el menú Apple, abra Preferencias del sistema.
2. Seleccione Seguridad y privacidad, a continuación, en la pestaña General, haga clic en el icono del candado para realizar cambios en la configuración de seguridad.
3. En el caso de devicetester_mac_x86-64 bloqueado, busque el mensaje "devicetester_mac_x86-64" was blocked from use because it is not from an identified developer. y elija Permitir de todos modos.
4. Reanude las pruebas de IDT hasta que haya revisado todos los ejecutables involucrados.

Errores de conexión SSH

Cuando IDT no se puede conectar a un dispositivo a prueba, los errores de conexión se registran en /results/*<execution-id>*/logs/*<test-case-id>*.log. Los mensajes de SSH aparecen en

la parte superior de este archivo de registro ya que la conexión a un dispositivo bajo prueba es una de las primeras operaciones que realiza IDT.

La mayoría de las configuraciones de Windows utilizan la aplicación de TTy terminal Pu para conectarse a los hosts Linux. Esta aplicación requiere que los archivos de clave privada PEM estándar se conviertan en un formato propio de Windows denominado PPK. Si configura SSH en su archivo `device.json`, utilice archivos PEM. Si utilizas un archivo PPK, IDT no puede crear una conexión SSH con el AWS IoT Greengrass dispositivo ni puede ejecutar pruebas.

A partir de la versión 4.4.0 de IDT, si no ha activado el SFTP en el dispositivo que está probando, es posible que vea el siguiente error en el archivo de registro.

```
SSH connection failed with EOF
```

Para corregir este error, active SFTP en el dispositivo.

Errores de calificación del administrador de flujos

Al ejecutar las pruebas de calificación del administrador de flujos, es posible que vea el siguiente error en el archivo `com.aws.StreamManagerExport.log`.

```
Failed to upload data to S3
```

Este error puede producirse cuando el administrador de transmisiones usa las AWS credenciales del `~/root/.aws/credentials` archivo del dispositivo en lugar de usar las credenciales de entorno que IDT exporta al dispositivo que se está probando. Para evitar este problema, elimine el archivo `credentials` de su dispositivo y vuelva a ejecutar la prueba de calificación.

Errores de tiempo de espera

Puede aumentar el tiempo de espera de cada prueba al especificar un multiplicador de tiempo de espera, que se aplicará al valor predeterminado del tiempo de espera de cada prueba. Cualquier valor configurado para esta marca debe ser superior o igual a 1,0.

Para utilizar el multiplicador de tiempo de espera, utilice la marca `--timeout-multiplier` al ejecutar las pruebas. Por ejemplo:

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

Para obtener más información, ejecute `run-suite --help`.

Algunos errores de tiempo de espera se producen cuando los casos de prueba de IDT no se pueden completar debido a problemas de configuración. No puede resolver estos errores aumentando el multiplicador de tiempo de espera. Use los registros de la ejecución de la prueba para solucionar los problemas de configuración subyacentes.

- Si los registros de los componentes MQTT o Lambda contienen errores `Access denied`, es posible que la carpeta de instalación de Greengrass no tenga los permisos de archivo correctos. Ejecute el siguiente comando para cada carpeta de la ruta de instalación que usted definió en su archivo `userdata.json`.

  ```
  sudo chmod 755 folder-name
  ```

- Si los registros de Greengrass indican que la implementación de la CLI de Greengrass no está completa, haga lo siguiente:
  - Compruebe que `bash` está instalado en el dispositivo que se está probando.
  - Si el archivo `userdata.json` incluye el parámetro de configuración `GreengrassCliVersion`, elimínelo. Esta característica quedó obsoleto en IDT versión 4.1.0 o en versiones posteriores. Para obtener más información, consulte Configuración de userdata.json.
- Si la prueba de implementación de Lambda ha fallado y aparece el mensaje de error “Validación de la publicación de Lambda: se ha agotado el tiempo de espera” y recibe un error en el archivo de registro de pruebas (`idt-gg2-lambda-function-idt-<resource-id>.log`) que dice `Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.`, haga lo siguiente:
  - Compruebe para qué carpeta se utilizó `InstallationDirRootOnDevice` en el archivo `userdata.json`.
  - Asegúrese de que los permisos de usuario correctos estén configurados en su dispositivo. Para obtener más información, consulte Configuración de los permisos de usuario en el dispositivo.

Errores de comprobación de versiones

IDT emite el siguiente error cuando las credenciales de AWS usuario del usuario de IDT no tienen los permisos de IAM necesarios.

```
Failed to check version compatibility
```

El AWS usuario que no tiene los permisos de IAM necesarios.

Política de soporte de AWS IoT Device Tester para AWS IoT Greengrass

AWS IoT Device Tester para AWS IoT Greengrass es una herramienta de automatización de pruebas que se utiliza para validar y comprobar si los dispositivos de AWS IoT Greengrass cumplen los requisitos para su inclusión en el Catálogo de dispositivos de AWS Partner. Es conveniente que utilice la versión más reciente de AWS IoT Greengrass y de AWS IoT Device Tester para probar o calificar los dispositivos.

Hay al menos una versión de AWS IoT Device Tester disponible para cada versión compatible de AWS IoT Greengrass. Para ver las versiones compatibles de AWS IoT Greengrass, consulte las versiones del núcleo de Greengrass. Para ver las versiones compatibles de AWS IoT Device Tester, consulte Versiones compatibles de AWS IoT Device Tester para AWS IoT Greengrass V2.

También puede utilizar cualquiera de las versiones compatibles de AWS IoT Greengrass e AWS IoT Device Tester para probar los dispositivos o comprobar si cumplen los requisitos. Aunque puede seguir utilizando versiones no compatibles de AWS IoT Device Tester, dichas versiones no reciben actualizaciones ni correcciones de errores. Si tiene alguna pregunta acerca de la política de compatibilidad, póngase en contacto con AWS Support.

Soluciones de IoT basadas en Greengrass

Everyware GreenEdge de Eurotech se encuentra en versión preliminar para AWS IoT Greengras s, por lo que está sujeto a cambios. no es compatible con esta solución AWS. Si tiene algún problema con este dispositivo, contáctese con Eurotech.

AWS IoT Greengrass ofrece soluciones de Socios para optimizar la experiencia de instalación de Greengrass. La siguiente es una solución que ofrece AWS en asociación con Eurotech. Esta solución viene con el tiempo de ejecución de periferia de AWS IoT Greengrass Core y capacidades adicionales preinstaladas.

Eurotech

AWS se asoció con Eurotech para ofrecer una solución de IoT a los clientes que buscan un dispositivo con el software AWS IoT Greengrass Core preinstalado. Everyware GreenEdge de Eurotech es un software de periferia de IoT preconfigurado y precalificado por AWS. Esta solución combina las capacidades de Greengrass y el Eurotech Everyware Software Framework (ESF) para ofrecer a los clientes una amplia conectividad hacia el sur mediante adaptadores de protocolo como: Modbus, OPC-UA Client/Server, S7, TwinCat, J1939, DNP3 Master/Outstation y más. Con esta solución, también puede enviar datos a Nube de AWS y conectarlo a todos los servicios de AWS en dirección norte (como AWS IoT Core, AWS IoT SiteWise, AWS IoT Analytics, Amazon S3 y Amazon Kinesis Video Streams). En combinación con Everyware Cloud, la solución de administración de dispositivos de Eurotech, esta solución presenta un novedoso servicio de aprovisionamiento sin intervención previa, que simplifica la incorporación y la implementación masiva de dispositivos.

Para obtener más información sobre Eurotech, consulte Eurotech.

Solución de problemas AWS IoT Greengrass V2

Utilice la información y las soluciones de solución de problemas de esta sección para ayudar a resolver los problemas con. AWS IoT Greengrass Version 2

Temas

- Consulte los registros AWS IoT Greengrass principales de software y componentes
- AWS IoT Greengrass Problemas principales de software
- AWS IoT Greengrass problemas con la nube
- Problemas de implementación del dispositivo principal
- Problemas con los componentes del dispositivo principal
- Problemas con los componentes de la función de Lambda del dispositivo principal
- Versión del componente descontinuada
- Problemas con la interfaz de la línea de comandos de Greengrass
- AWS Command Line Interface problemas
- Códigos de error de implementación detallados
- Códigos de estado de componentes detallados

Consulte los registros AWS IoT Greengrass principales de software y componentes

El software AWS IoT Greengrass Core escribe registros en el sistema de archivos local que puede usar para ver información en tiempo real sobre el dispositivo principal. También puede configurar los dispositivos principales para que escriban registros en los CloudWatch registros, de modo que pueda solucionar los problemas de los dispositivos principales de forma remota. Estos registros pueden ayudarlo a identificar problemas con los componentes, las implementaciones y los dispositivos principales. Para obtener más información, consulte Supervisión de los registros de AWS IoT Greengrass.

AWS IoT Greengrass Problemas principales de software

Solucionar problemas AWS IoT Greengrass de software principal.

Temas

- ThrottlingException desde ListDeployments la API
- No se ha podido configurar el dispositivo principal
- No se puede iniciar el software AWS IoT Greengrass Core como un servicio del sistema
- No se puede configurar el núcleo como un servicio del sistema
- No se puede conectar a AWS IoT Core
- Error de falta de memoria
- No se puede instalar la CLI de Greengrass
- User root is not allowed to execute
- com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with
- Failed to map segment from shared object: operation not permitted
- No se pudo configurar el servicio de Windows
- com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
- com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime
- software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid
- software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy
- Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request
- Operation aws.greengrass#<operation> is not supported by Greengrass
- java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed

- java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED
- Greengrass core device stuck on nucleus v2.12.3
- Greengrass nucleus v2.14.0 systemd template issue

ThrottlingException desde ListDeployments la API

`ThrottlingException` de la API `ListDeployments`: es posible que observe esto cuando tenga una gran cantidad de implementaciones.

Para resolver este problema, siga uno de estos pasos:

- Si usa el SDK, especifique el MaxResult parámetro. Por ejemplo, para JavaSDK con un valor pequeño (por ejemplo, 5).
- Puede usar AWS Service Quotas para solicitar un aumento del límite de velocidad de la API `DescribeJob`. Puedes ir a la consola de cuotas de servicio, seleccionar las cuotas AWS IoT y el nombre del límite es DescribeJob throttle limit. Puede aumentarlo de 10 a 50.

No se ha podido configurar el dispositivo principal

Si el instalador del software AWS IoT Greengrass principal falla y no puedes configurar un dispositivo principal, es posible que tengas que desinstalar el software y volver a intentarlo. Para obtener más información, consulte Desinstalación del software AWS IoT Greengrass Core.

No se puede iniciar el software AWS IoT Greengrass Core como un servicio del sistema

Si el software AWS IoT Greengrass principal no se inicia, compruebe los registros de servicio del sistema para identificar el problema. Un problema habitual es que Java no esté disponible en la variable de entorno PATH (Linux) o en la variable de sistema PATH (Windows).

No se puede configurar el núcleo como un servicio del sistema

Es posible que aparezca este error si el instalador del software AWS IoT Greengrass principal no se configura AWS IoT Greengrass como un servicio del sistema. En los dispositivos Linux, este error

suele producirse si el dispositivo principal no tiene el sistema de inicio systemd. El instalador puede configurar correctamente el software AWS IoT Greengrass principal aunque no pueda configurar el servicio del sistema.

Realice una de las siguientes acciones:

- Configure y ejecute el software AWS IoT Greengrass principal como un servicio del sistema. Debe configurar el software como un servicio del sistema para poder utilizar todas las características de AWS IoT Greengrass. Puede instalar systemd o utilizar un sistema de inicio diferente. Para obtener más información, consulte Configuración del núcleo de Greengrass como un servicio del sistema.
- Ejecute el software AWS IoT Greengrass principal sin un servicio de sistema. Puede ejecutar el software mediante un script de cargador que el instalador configura en la carpeta raíz de Greengrass. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass Core sin un servicio de sistema.

No se puede conectar a AWS IoT Core

Es posible que aparezca este error cuando el software AWS IoT Greengrass principal no se pueda conectar AWS IoT Core para recuperar los trabajos de implementación, por ejemplo. Haga lo siguiente:

- Comprueba que tu dispositivo principal se pueda conectar a Internet y AWS IoT Core. Para obtener más información sobre el AWS IoT Core punto final al que se conecta el dispositivo, consulteConfigurar el software AWS IoT Greengrass principal.
- Comprueba que el dispositivo AWS IoT principal utilice un certificado que permita los `iot:Subscribe` permisos `iot:Connect` `iot:Publish``iot:Receive`,, y.
- Si su dispositivo principal usa un proxy de red, compruebe que el dispositivo principal tenga un rol de dispositivo y que este rol conceda los permisos `iot:Connect`, `iot:Publish`, `iot:Receive` y `iot:Subscribe`.

Error de falta de memoria

Este error suele producirse si el dispositivo no tiene memoria suficiente para asignar un objeto en el montón de Java. En los dispositivos con memoria limitada, es posible que tenga que especificar un tamaño máximo de pila para controlar la asignación de memoria. Para obtener más información, consulte Control de la asignación de memoria con las opciones de JVM.

No se puede instalar la CLI de Greengrass

Es posible que veas el siguiente mensaje de consola cuando utilices el `--deploy-dev-tools` argumento en el comando de instalación de AWS IoT Greengrass Core.

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
 deployment for Greengrass first party dev tools, please manually create a deployment
 if you wish to
```

Esto ocurre cuando el componente CLI de Greengrass no está instalado porque el dispositivo principal es miembro de un grupo de objetos que tiene una implementación existente. Si ve este mensaje, puede implementar manualmente el componente CLI de Greengrass (`aws.greengrass.Cli`) en el dispositivo para instalar la CLI de Greengrass. Para obtener más información, consulte Instalación de la CLI de Greengrass.

User root is not allowed to execute

Es posible que aparezca este error cuando el usuario que ejecuta el software AWS IoT Greengrass Core (normalmente`root`) no tiene permiso para ejecutar `sudo` con ningún usuario ni grupo. Para el usuario `ggc_user` predeterminado del sistema, este error tiene el siguiente aspecto:

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

Compruebe que el archivo `/etc/sudoers` da permiso al usuario para ejecutar `sudo` como otros grupos. El permiso para el usuario en `/etc/sudoers` debería verse como el siguiente ejemplo.

```
root    ALL=(ALL:ALL) ALL
```

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

Es posible que aparezca este error cuando el dispositivo principal intente ejecutar un componente y el núcleo de Greengrass no especifique un usuario de sistema predeterminado para ejecutar los componentes.

Para solucionar este problema, configure el núcleo de Greengrass para especificar el usuario del sistema predeterminado que ejecuta los componentes. Para obtener más información, consulte Configuración del usuario que ejecuta los componentes y Configuración del usuario del componente predeterminado.

Failed to map segment from shared object: operation not permitted

Es posible que veas este error cuando el software AWS IoT Greengrass principal no se inicie porque la `/tmp` carpeta está montada con `noexec` permisos. La biblioteca Common Runtime (CRT) de AWS usa la carpeta `/tmp` de forma predeterminada.

Realice una de las siguientes acciones:

- Ejecute el siguiente comando para volver a montar la carpeta `/tmp` con permisos `exec` e inténtelo de nuevo.

```
sudo mount -o remount,exec /tmp
```

- Si ejecuta Greengrass nucleus v2.5.0 o posterior, puede configurar una opción de JVM para cambiar la carpeta que utiliza la biblioteca CRT. AWS Puede especificar el `jvmOptions` parámetro en la configuración del componente núcleo de Greengrass en una implementación o al instalar el software AWS IoT Greengrass Core. *`/path/to/use`*Sustitúyalo por la ruta a una carpeta que pueda usar la biblioteca AWS CRT.

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

No se pudo configurar el servicio de Windows

Es posible que veas este error si instalas el software AWS IoT Greengrass Core en un dispositivo Microsoft Windows 2016. El software AWS IoT Greengrass principal no es compatible con Windows 2016; para obtener una lista de los sistemas operativos compatibles, consultePlataformas admitidas.

Si debe utilizar Windows 2016, puede hacer lo siguiente:

1. Descomprima el archivo de instalación AWS IoT Greengrass de Core descargado
2. Abra el archivo `bin/greengrass.xml.template` en el directorio `Greengrass`.
3. Agregue la etiqueta `<autoRefresh>` al final del archivo justo antes de la etiqueta `</service>`.

```
  </log>
  <autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

Es posible que aparezca este error al instalar el software AWS IoT Greengrass Core sin un archivo raíz de una entidad emisora de certificados (CA).

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
 service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
 com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

Compruebe que ha especificado un archivo de CA raíz válido con el parámetro `rootCaPath` en el archivo de configuración que ha proporcionado al instalador. Para obtener más información, consulte Instalación del software AWS IoT Greengrass Core.

com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime

Es posible que veas este mensaje de advertencia cuando el dispositivo principal no pueda conectarse para suscribirse AWS IoT Core a las notificaciones de tareas de implementación. Haga lo siguiente:

- Compruebe que el dispositivo principal esté conectado a Internet y pueda acceder al punto final de AWS IoT datos que configuró. Para obtener más información acerca de los puntos de conexión que utilizan los dispositivos principales, consulte Cómo permitir el tráfico del dispositivo a través de un proxy o firewall.
- Compruebe los registros de Greengrass para ver si hay otros errores que revelen otras causas principales.

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

Es posible que aparezca este error al instalar el software AWS IoT Greengrass principal con el aprovisionamiento automático y el instalador utilice un token de AWS sesión que no es válido. Haga lo siguiente:

- Si utiliza credenciales de seguridad temporales, compruebe que el token de sesión es correcto y que está copiando y pegando el token de sesión completo.
- Si utiliza credenciales de seguridad de larga duración, compruebe que el dispositivo no tenga un token de sesión de una época en la que utilizó credenciales temporales. Haga lo siguiente:
  1. Ejecute el siguiente comando para anular la configuración de la variable de entorno del token de sesión.

     Linux or Unix

     ```
     unset AWS_SESSION_TOKEN
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_SESSION_TOKEN=
     ```

     PowerShell

     ```
     Remove-Item Env:\AWS_SESSION_TOKEN
     ```

  2. Compruebe si el archivo de AWS credenciales,`~/.aws/credentials`, contiene un token de sesión,`aws_session_token`. Si es así, elimine esa línea del archivo.

     ```
     aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
     +FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
     IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
     ```

También puede instalar el software AWS IoT Greengrass Core sin proporcionar AWS credenciales. Para obtener más información, consulte Instale el software AWS IoT Greengrass principal con aprovisionamiento manual de recursos o Instale el software AWS IoT Greengrass principal con aprovisionamiento AWS IoT de flota.

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

Es posible que aparezca este error al instalar el software AWS IoT Greengrass principal con el aprovisionamiento automático y el instalador utilice AWS credenciales que no tienen los permisos necesarios. Para obtener más información sobre los permisos necesarios, consulte Política de IAM mínima para que el instalador aprovisione recursos.

Compruebe los permisos de la identidad de IAM de las credenciales y otorgue a la identidad de IAM los permisos necesarios que falten.

Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request

Es posible que veas este error cuando utilices el componente administrador de sombras para sincronizar dispositivos ocultos con AWS IoT Core ellos. El código de estado HTTP 403 indica que este error se produjo porque la AWS IoT política del dispositivo principal no concede permiso para realizar llamadas`GetThingShadow`.

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
 cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
 com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
 name=MyGreengrassCore, shadow name=MyShadow}
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
 software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
 null (Service: IotDataPlane, Status Code: 403, Request ID:
 f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

Para sincronizar con las sombras locales AWS IoT Core, la AWS IoT política del dispositivo principal debe conceder los siguientes permisos:

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Comprueba la AWS IoT política del dispositivo principal y añade los permisos necesarios que falten. Para obtener más información, consulte los siguientes temas:

- AWS IoT Core acciones políticas en la Guía para AWS IoT desarrolladores
- Actualice la AWS IoT política de un dispositivo principal

Operation aws.greengrass#<operation> is not supported by Greengrass

Es posible que aparezca este error cuando utilice una operación de comunicación entre procesos (IPC) en un componente personalizado de Greengrass y el componente requerido AWS proporcionado no esté instalado en el dispositivo principal.

Para solucionar este problema, añada el componente necesario como una dependencia en la receta de componentes, de modo que el software AWS IoT Greengrass principal instale el componente necesario al implementar el componente.

- Recupera los valores secretos: `aws.greengrass.SecretManager`
- Interactúa con las sombras locales: `aws.greengrass.ShadowManager`
- Administre las implementaciones y los componentes locales: `aws.greengrass.Cli` versión 2.6.0 o posterior
- Autentice y autorice los dispositivos DE cliente: `aws.greengrass.clientdevices.Auth` versión 2.2.0 o posterior

java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)

Es posible que vea este error en el archivo de registro del administrador de flujos (`aws.greengrass.StreamManager.log`) cuando configura el administrador de flujos para que use una carpeta raíz que no existe o que no tiene los permisos correctos. Para obtener más información sobre cómo configurar esta carpeta, consulte configuración del administrador de flujos.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

Este error se produce cuando el componente proveedor PKCS #11 no encuentra ni carga la clave privada o el certificado que especificó al configurar el software AWS IoT Greengrass principal para que utilice un módulo de seguridad de hardware (HSM). Haga lo siguiente:

- Compruebe que la clave privada y el certificado estén almacenados en el HSM mediante la ranura, el PIN de usuario y la etiqueta de objeto para los que ha configurado el software AWS IoT Greengrass Core.
- Compruebe que la clave privada y el certificado utilizan la misma etiqueta de objeto en el HSM.

- Si su HSM admite un objeto IDs, compruebe que la clave privada y el certificado utilicen el mismo ID de objeto en el HSM.

Consulte la documentación de su HSM para obtener información sobre cómo consultar los detalles sobre los tokens de seguridad del HSM. Si necesita cambiar la ranura, la etiqueta del objeto o el identificador del objeto por un token de seguridad, consulte la documentación de su HSM para obtener información sobre cómo hacerlo.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

Este error puede producirse cuando se utiliza el componente de administrador de secretos para implementar un AWS Secrets Manager secreto. Si el rol de IAM de intercambio de token del dispositivo principal no otorga permiso para obtener el secreto, la implementación falla y los registros de Greengrass incluyen este error.

Cómo autorizar a un dispositivo principal a descargar un secreto

1. Agregue el permiso `secretsmanager:GetSecretValue` al rol de intercambio de token del dispositivo principal. En el siguiente ejemplo de declaración de política, se concede permiso para obtener el valor de un secreto.

   ```
   {
       "Effect": "Allow",
       "Action": [
           "secretsmanager:GetSecretValue"
       ],
       "Resource": [
           "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
   abcdef"
       ]
   }
   ```

   Para obtener más información, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

2. Vuelve a aplicar la implementación al dispositivo principal. Realice una de las siguientes acciones:

- Revise la implementación sin cambios. El dispositivo principal intenta volver a descargar el secreto cuando recibe la implementación revisada. Para obtener más información, consulte Revisión de las implementaciones.
- Reinicie el software AWS IoT Greengrass principal para volver a intentar la implementación. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal

La implementación se realiza correctamente si el administrador de secretos descarga el secreto correctamente.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep Access to KMS is not allowed

Este error puede producirse cuando se utiliza el componente de administrador de secretos para implementar un AWS Secrets Manager secreto cifrado mediante una AWS Key Management Service clave. Si el rol de IAM de intercambio de token del dispositivo principal no otorga permiso para descifrar el secreto, la implementación falla y los registros de Greengrass incluyen este error.

Para solucionar el problema, agregue el permiso `kms:Decrypt` al rol de intercambio de token del dispositivo principal. Para obtener más información, consulte los siguientes temas:

- Cifrado y descifrado de secretos en la Guía del usuario de AWS Secrets Manager
- Autorizar a los dispositivos principales a interactuar con AWS los servicios

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

Es posible que veas este error cuando intentes instalar el software AWS IoT Greengrass Core con seguridad de hardware y utilices una versión anterior del núcleo de Greengrass que no admite la integración de seguridad de hardware. Para usar la integración de seguridad de hardware, debe usar el núcleo de Greengrass versión 2.5.3 o posterior.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

Es posible que aparezca este error cuando utilice la TPM2 biblioteca cuando ejecute AWS IoT Greengrass Core como un servicio del sistema.

Este error indica que debe agregar una variable de entorno que proporcione la ubicación del almacén PKCS #11 en el archivo de servicio AWS IoT Greengrass Core systemd.

Para obtener más información, consulte la sección Requisitos de la documentación del componente Proveedor PKCS#11.

Greengrass core device stuck on nucleus v2.12.3

Si su dispositivo principal de Greengrass no revisa la implementación de la versión 2.12.3 de núcleo, es posible que tenga que descargar y reemplazar el archivo `Greengrass.jar` por la versión 2.12.2 del núcleo de Greengrass. Haga lo siguiente:

1. Ejecute el siguiente comando en su dispositivo principal de Greengrass para detener el software de Greengrass Core.

   Linux or Unix

   ```
   sudo systemctl stop greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc stop "greengrass"
   ```

   PowerShell

   ```
   Stop-Service -Name "greengrass"
   ```

2. En su dispositivo principal, descargue el AWS IoT Greengrass software a un archivo denominado. `greengrass-2.12.2.zip`

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -
OutFile greengrass-2.12.2.zip
```

3. Descomprime el software AWS IoT Greengrass principal en una carpeta de tu dispositivo. *GreengrassInstaller*Sustitúyalo por la carpeta que desee usar.

   Linux or Unix

   ```
   unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -
   C GreengrassInstaller && del greengrass-2.12.2.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-2.12.2.zip
   ```

4. Ejecute el siguiente comando para anular el archivo JAR de Greengrass de la versión 2.12.3 del núcleo por el archivo JAR de Greengrass de la versión 2.12.2 del núcleo.

   Linux or Unix

   ```
   sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
   ```

PowerShell

```
cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /
greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/
aws.greengrass.nucleus/lib
```

5. Ejecute el siguiente comando para iniciar el software de Greengrass Core.

   Linux or Unix

   ```
   sudo systemctl start greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc start "greengrass"
   ```

   PowerShell

   ```
   Start-Service -Name "greengrass"
   ```

Greengrass nucleus v2.14.0 systemd template issue

Es posible que se encuentre con este problema si ha instalado la versión 2.14.0 del núcleo de Greengrass en un dispositivo Linux con su plantilla de servicio systemd predeterminada. Haga lo siguiente:

1. En su dispositivo principal de Greengrass, ejecute el siguiente comando para revertir el archivo de servicio de systemd a la forma en que estaba en la versión 2.13.0 y versiones anteriores del núcleo.

   Linux or Unix

   ```
   sudo sed -i 's|ExecStart=/bin/sh -c "\(.*\) >> .*/logs/loader.log 2>&1"|
   ExecStart=/bin/sh \1|' /etc/systemd/system/greengrass.service
   ```

2. Aplique los cambios.

Linux or Unix

```
sudo systemctl daemon-reload
sudo systemctl restart greengrass
```

AWS IoT Greengrass problemas con la nube

Usa la siguiente información para solucionar problemas con la AWS IoT Greengrass consola y la API. Cada entrada corresponde a un mensaje de error que puede aparecer al realizar una acción.

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

Es posible que aparezca este error al crear una versión de componente desde la AWS IoT Greengrass consola o durante la CreateComponentVersionoperación.

Este error indica que su receta no es válida en formato JSON o YAML. Compruebe la sintaxis de su receta, corrija cualquier problema de sintaxis e inténtelo de nuevo. Puede usar un corrector de sintaxis JSON o YAML en línea para identificar los problemas de sintaxis en su receta.

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

Es posible que aparezca este error al crear una versión de componente desde la AWS IoT Greengrass consola o durante la CreateComponentVersionoperación. Este error indica que un artefacto de S3 en la receta del componente no es válido.

Haga lo siguiente:

- Compruebe que el depósito de S3 esté en el mismo Región de AWS lugar en el que creó el componente. AWS IoT Greengrass no admite solicitudes de artefactos de componentes entre regiones.
- Compruebe que el URI del artefacto es una URL de objeto S3 válida y compruebe que el artefacto existe en esa URL de objeto S3.

- Comprueba que tienes Cuenta de AWS permiso para acceder al artefacto en la URL del objeto S3.

INACTIVE deployment status

Es posible que obtengas un estado de `INACTIVE` implementación cuando llames a la ListDeploymentsAPI sin las AWS IoT políticas dependientes requeridas. Debe tener los permisos necesarios para obtener un estado de implementación preciso. Puede encontrar las acciones dependientes consultando las acciones definidas por AWS IoT Greengrass V2 y siguiendo los permisos necesarios para `ListDeployments`. Sin los AWS IoT permisos dependientes necesarios, seguirás viendo el estado de la implementación, pero es posible que veas un estado de implementación incorrecto de`INACTIVE`.

Problemas de implementación del dispositivo principal

Solución de problemas en las implementaciones de los dispositivos principales de Greengrass. Cada entrada corresponde a un mensaje de registro que puede que vea en su dispositivo principal.

Temas

- Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact
- Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.
- Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>
- software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
- com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component
- Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

- Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration
- Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy
- Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration
- Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadExceptio Failed to download artifact

Es posible que aparezca este error cuando el software AWS IoT Greengrass principal no puede descargar un artefacto componente cuando el dispositivo principal realiza una implementación. La implementación falla como resultado de este error.

Cuando recibe este error, el registro también incluye un rastreo de pila que puede usar para identificar el problema específico. Cada una de las siguientes entradas corresponde a un mensaje que puede que vea en el rastreo de pila del mensaje de error `Failed to download artifact`.

Temas

- software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)
- software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

El PackageDownloadException error puede incluir este rastreo de pila en los siguientes casos:

- El artefacto del componente no está disponible en la URL del objeto de S3 que especificó en la receta del componente. Compruebe que ha subido el artefacto al bucket de S3 y que el URI del artefacto coincide con la URL del objeto de S3 del artefacto en el bucket.
- La función de intercambio de fichas del dispositivo principal no permite que el software AWS IoT Greengrass principal descargue el artefacto componente desde la URL del objeto S3 que se especifique en la receta del componente. Compruebe que la función de intercambio de token permita `s3:GetObject` de la URL del objeto de S3 en la que el artefacto está disponible.

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

El PackageDownloadException error puede incluir este seguimiento de pila cuando el dispositivo principal no tiene permiso para llamars`3:GetBucketLocation`. El mensaje de error incluye uno de los siguientes mensajes.

```
reason: Failed to determine S3 bucket location
```

Compruebe que el rol de intercambio de token permita `s3:GetBucketLocation` para el bucket de S3 en la que el artefacto está disponible.

Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchI Integrity check for downloaded artifact failed. Probably due to file corruption.

Es posible que aparezca este error cuando el software AWS IoT Greengrass principal no puede descargar un artefacto componente cuando el dispositivo principal realiza una implementación. La implementación falla porque la suma de verificación del archivo de artefactos descargado no coincide con la suma de verificación que AWS IoT Greengrass se calculó al crear el componente.

Haga lo siguiente:

- Compruebe si el archivo de artefactos ha cambiado en el bucket de S3 donde lo aloja. Si el archivo ha cambiado desde que creó el componente, restáurelo a la versión anterior que esperaba el dispositivo principal. Si no puede restaurar el archivo a su versión anterior o si desea utilizar la nueva versión del archivo, cree una nueva versión del componente con el archivo de artefactos.

- Compruebe la conexión a Internet del dispositivo principal. Este error puede producirse si el archivo del artefacto se daña mientras se descarga. Cree una implementación nueva e inténtelo de nuevo.

Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVers Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

Es posible que aparezca este error cuando un dispositivo principal no encuentre una versión de componente que cumpla con los requisitos de las implementaciones de ese dispositivo principal. El dispositivo principal comprueba el componente en el AWS IoT Greengrass servicio y en el dispositivo local. El mensaje de error incluye el destino de cada implementación y los requisitos de versión de esa implementación para el componente. El destino de la implementación puede ser un objeto, un grupo de objetos o `LOCAL_DEPLOYMENT`, que represente la implementación local en el dispositivo principal.

Este error se puede producir en los siguientes escenarios:

- El dispositivo principal es el objetivo de varias implementaciones que tienen requisitos de versión de componentes contradictorios. Por ejemplo, el dispositivo principal puede ser el objetivo de varias implementaciones que incluyen un componente `com.example.HelloWorld`, en las que una implementación requiere la versión 1.0.0 y la otra requiere la versión 1.0.1. Es imposible tener un componente que cumpla ambos requisitos, por lo que la implementación falla.
- La versión del componente no existe en el AWS IoT Greengrass servicio ni en el dispositivo local. Es posible que el componente se haya eliminado, por ejemplo.
- Existen versiones de componentes que cumplen con los requisitos de la versión, pero ninguna es compatible con la plataforma del dispositivo principal.
- La AWS IoT política del dispositivo principal no concede el `greengrass:ResolveComponentCandidates` permiso. Busque `Status Code: 403` en el registro de errores para identificar este problema. Para resolver este problema, agregue el permiso `greengrass:ResolveComponentCandidates` a la política AWS IoT del dispositivo principal. Para obtener más información, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales.

Para resolver este problema, revise las implementaciones para incluir versiones de componentes compatibles o eliminar los incompatibles. Para obtener más información sobre cómo revisar las implementaciones en la nube, consulte Revisión de las implementaciones. Para obtener información sobre cómo revisar implementaciones locales, consulte el comando Crear implementaciones de la CLI de AWS IoT Greengrass.

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundE The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

Es posible que aparezca este error al implementar un componente en un dispositivo principal y el componente no muestre una plataforma que sea compatible con la plataforma del dispositivo principal. Realice una de las siguientes acciones:

- Si el componente es un componente personalizado de Greengrass, puede actualizarlo para que sea compatible con el dispositivo principal. Agregue un nuevo manifiesto que coincida con la plataforma del dispositivo principal o actualice un manifiesto existente para que coincida con la plataforma del dispositivo principal. Para obtener más información, consulte AWS IoT Greengrass referencia de recetas de componentes.
- Si el componente lo proporciona AWS, compruebe si otra versión del componente es compatible con el dispositivo principal. Si ninguna versión es compatible, póngase en contacto con nosotros mediante AWS re:Post usando la etiqueta AWS IoT Greengrass o póngase en contacto con Soporte.

com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component

Es posible que aparezca este error al implementar un componente que depende del núcleo de Greengrass y el dispositivo principal ejecute una versión del núcleo de Greengrass anterior a la última versión secundaria disponible. Este error se produce porque el software AWS IoT Greengrass principal intenta actualizar automáticamente los componentes a la última versión compatible. Sin embargo, el software AWS IoT Greengrass Core impide que el núcleo de Greengrass se actualice a una nueva versión secundaria, ya que varios de los componentes AWS proporcionados dependen

de versiones secundarias específicas del núcleo de Greengrass. Para obtener más información, consulte Comportamiento de actualización del núcleo de Greengrass.

Debe revisar la implementación para especificar la versión del núcleo de Greengrass que quiere usar. Realice una de las siguientes acciones:

- Revise la implementación para especificar la versión del núcleo de Greengrass que ejecuta actualmente el dispositivo principal.
- Revise la implementación para especificar una versión secundaria posterior del núcleo de Greengrass. Si elige esta opción, también debe actualizar las versiones de todos los componentes AWS proporcionados que dependen de versiones secundarias específicas del núcleo de Greengrass. Para obtener más información, consulte Componentes proporcionados por AWS.

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

Es posible que vea este error cuando mueva un dispositivo de Greengrass de un grupo de objetos a otro y luego de vuelta al grupo original con implementaciones que requieren que Greengrass se reinicie.

Para resolver este problema, vuelva a crear el directorio de inicio del dispositivo. También recomendamos encarecidamente actualizar el núcleo de Greengrass a la versión 2.9.6 o posterior.

El siguiente es un script de Linux para recrear el directorio de inicio. Guarde el script en un archivo denominado `fix_directory.sh`.

```
#!/bin/bash

set -e

GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
  mkdir -p $GG_ROOT/alts/directory_fix
  echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
```

```
  ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
  echo "Creating directory: $TARGET"
  mkdir -p "$TARGET"
fi

DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

Para ejecutar el script, ejecute el siguiente comando:

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDow Greengrass Cloud Service returned an error when getting full deployment configuration

Es posible que aparezca este error cuando el dispositivo principal reciba un documento de implementación de gran tamaño, es decir, un documento de implementación de más de 7 KB (en el caso de las implementaciones dirigidas a objetos) o 31 KB (en el caso de las implementaciones dirigidas a grupos de objetos). Para recuperar un documento de despliegue de gran tamaño, la AWS IoT política del dispositivo principal debe permitir el `greengrass:GetDeploymentConfiguration` permiso. Este error puede producirse cuando el dispositivo principal no tiene este permiso. Cuando se produce este error, la implementación se reintenta indefinidamente y su estado es En curso (`IN_PROGRESS`).

Para resolver este problema, añade el `greengrass:GetDeploymentConfiguration` permiso a la AWS IoT política del dispositivo principal. Para obtener más información, consulte Actualice la AWS IoT política de un dispositivo principal.

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

Es posible que veas esta advertencia cuando el dispositivo principal reciba una implementación y la AWS IoT política del dispositivo principal no permita el `greengrass:ListThingGroupsForCoreDevice` permiso. Cuando crea una implementación, el dispositivo principal usa este permiso para identificar sus grupos de objetos y eliminar componentes de cualquier grupo de objetos que haya eliminado del dispositivo principal. Si el dispositivo principal ejecuta el núcleo de Greengrass versión 2.5.0, se produce un error en la implementación. Si el dispositivo principal ejecuta el núcleo de Greengrass versión 2.5.1 o posterior, la implementación continúa, pero no elimina los componentes. Para obtener más información acerca de comportamientos de eliminación de grupos de objeto, consulte Implemente AWS IoT Greengrass componentes en los dispositivos.

Para actualizar el comportamiento del dispositivo principal a fin de eliminar componentes de los grupos de cosas de los que se elimina el dispositivo principal, añada el `greengrass:ListThingGroupsForCoreDevice` permiso a la AWS IoT política del dispositivo principal. Para obtener más información, consulte Actualice la AWS IoT política de un dispositivo principal.

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

Es posible que vea este mensaje de información impreso varias veces sin ningún error, ya que el dispositivo principal registra el error a nivel de registro `DEBUG`. Este problema puede producirse cuando el dispositivo principal recibe un documento de implementación de gran tamaño. Cuando se produce este error, la implementación se reintenta indefinidamente y su estado es En curso (`IN_PROGRESS`). Para obtener más información sobre cómo resolver este problema, consulte esta entrada de solución de problemas.

Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataE null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Es posible que se muestre este error cuando la API de un plano de datos no tiene el permiso `iot:Connect`. Si no tiene la política correcta, recibirá un `GreengrassV2DataException: 403`. Para crear una política de permisos, siga estas instrucciones: Cree una AWS IoT política.

Problemas con los componentes del dispositivo principal

Solución de problemas en los componentes de los dispositivos principales de Greengrass.

Temas

- Warn: '<command>' is not recognized as an internal or external command
- El script de Python no registra los mensajes
- La configuración de los componentes no se actualiza al cambiar la configuración predeterminada
- awsiot.greengrasscoreipc.model.UnauthorizedError
- com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)
- com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers
- Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"
- copyFrom: <configurationPath> is already a container, not a leaf
- com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'
- java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.
- aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Warn: '<command>' is not recognized as an internal or external command

Es posible que veas este error en los registros de un componente de Greengrass cuando el software AWS IoT Greengrass principal no ejecute un comando en el script de ciclo de vida del componente. El estado del componente pasa a ser `BROKEN`, resultado de este error. Este error puede producirse si el usuario del sistema que ejecuta el componente, por ejemplo, `ggc_user`, no encuentra el ejecutable del comando en las carpetas PATH.

En los dispositivos Windows, compruebe que la carpeta que contiene el ejecutable pertenece a la `PATH` del sistema que ejecuta el componente. Si no aparece en la `PATH`, realice una de las siguientes opciones:

- Agregue la carpeta del ejecutable a la variable del sistema `PATH`, que está disponible para todos los usuarios. A continuación, reinicie el componente.

  Si ejecuta Greengrass nucleus 2.5.0, después de actualizar la variable de `PATH` sistema, debe reiniciar el software AWS IoT Greengrass Core para ejecutar los componentes con la actualización. `PATH` Si el software AWS IoT Greengrass principal no utiliza la actualización `PATH` después de reiniciar el software, reinicie el dispositivo e inténtelo de nuevo. Para obtener más información, consulte Ejecute el software AWS IoT Greengrass principal.
- Agregue la carpeta del ejecutable a la variable de usuario `PATH` del sistema que ejecuta el componente.

El script de Python no registra los mensajes

Los dispositivos principales de Greengrass recopilan registros que puede utilizar para identificar problemas con los componentes. Si los mensajes del script `stdout` y `stderr` de Python no aparecen en los registros de sus componentes, es posible que deba vaciar el búfer o deshabilitar el almacenamiento en búfer para estos flujos de salida estándar en Python. Realice uno de los siguientes procedimientos:

- Ejecute Python con el argumento -u para deshabilitar el almacenamiento en búfer en `stdout` y `stderr`.

  Linux or Unix

  ```
  python3 -u hello_world.py
  ```

Windows

```
py -3 -u hello_world.py
```

- Utilice Setenv en la receta de su componente para establecer la variable de entorno PYTHONUNBUFFERED en una cadena que no esté vacía. Esta variable de entorno deshabilita el almacenamiento en búfer en `stdout` y `stderr`.
- Vacíe el búfer de los flujos `stdout` o `stderr`. Realice una de las siguientes acciones:
  - Vacíe un mensaje al imprimirlo.

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

  - Vacíe un mensaje después de imprimirlo. Puede enviar varios mensajes antes de vaciar el flujo.

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

Para obtener más información acerca de cómo verificar que el script de Python genera mensajes de registro, consulte Supervisión de los registros de AWS IoT Greengrass.

La configuración de los componentes no se actualiza al cambiar la configuración predeterminada

Al cambiar la `DefaultConfiguration` en la receta de un componente, la nueva configuración predeterminada no sustituirá a la configuración existente del componente durante la implementación. Para aplicar la nueva configuración predeterminada, debe restablecer la configuración del componente a sus valores predeterminados. Al implementar el componente, especifique una sola cadena vacía como restablecimiento de actualización.

Console

Restablecer las rutas

```
[""]
```

AWS CLI

El siguiente comando crea una implementación a un dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-
deployment.json
```

El archivo `reset-configuration-deployment.json` contiene el siguiente documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {,
        "reset": [""]
      }
    }
  }
}
```

Greengrass CLI

El siguiente comando de la CLI de Greengrass crea una implementación local en un dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

El archivo `reset-configuration-deployment.json` contiene el siguiente documento JSON.

```
{
  "com.example.HelloWorld": {
    "RESET": [""]
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

Es posible que vea este error en los registros de un componente de Greengrass cuando el componente no tiene permiso para realizar una operación de IPC en un recurso. Para conceder permiso a un componente para llamar a una operación de IPC, defina una política de autorización de IPC en la configuración del componente. Para obtener más información, consulte Autorización de los componentes para realizar operaciones de IPC.

> **Tip**
>
> Si cambia `DefaultConfiguration` en la receta de un componente, debe restablecer la configuración del componente a su nueva configuración predeterminada. Al implementar el componente, especifique una sola cadena vacía como restablecimiento de actualización. Para obtener más información, consulte La configuración de los componentes no se actualiza al cambiar la configuración predeterminada.

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

Es posible que aparezca este error si varias políticas de autorización de IPC, incluidos todos los componentes del dispositivo principal, utilizan el mismo identificador de política.

Compruebe las políticas de autorización de IPC de sus componentes, corrija los duplicados e inténtelo de nuevo. Para crear una política única IDs, le recomendamos que combine el nombre del componente, el nombre del servicio de IPC y un contador. Para obtener más información, consulte Autorización de los componentes para realizar operaciones de IPC.

> **Tip**
>
> Si cambia `DefaultConfiguration` en la receta de un componente, debe restablecer la configuración del componente a su nueva configuración predeterminada. Al implementar el

componente, especifique una sola cadena vacía como restablecimiento de actualización. Para obtener más información, consulte La configuración de los componentes no se actualiza al cambiar la configuración predeterminada.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

Es posible que aparezca este error cuando un dispositivo principal no puede obtener AWS las credenciales del servicio de intercambio de fichas. El código de estado HTTP 400 indica que este error se produjo porque la función de IAM de intercambio de fichas del dispositivo principal no existe o no tiene una relación de confianza que permita al proveedor de AWS IoT credenciales asumirla.

Haga lo siguiente:

1. Identifique la función de intercambio de token que utiliza el dispositivo principal. El mensaje de error incluye el alias de la AWS IoT función del dispositivo principal, que apunta a la función de intercambio de fichas. Ejecuta el siguiente comando en tu ordenador de desarrollo y *MyGreengrassCoreTokenExchangeRoleAlias* sustitúyelo por el nombre del alias del AWS IoT rol que aparece en el mensaje de error.

   ```
   aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
   ```

   La respuesta incluye el nombre de recurso de Amazon (ARN) del rol de IAM de intercambio de token.

   ```
   {
     "roleAliasDescription": {
       "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
       "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
   MyGreengrassCoreTokenExchangeRoleAlias",
       "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
       "owner": "123456789012",
       "credentialDurationSeconds": 3600,
       "creationDate": "2021-02-05T16:46:18.042000-08:00",
       "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
     }
   }
   ```

2. Compruebe si el rol existe. Ejecuta el siguiente comando y *MyGreengrassV2TokenExchangeRole* sustitúyelo por el nombre del rol de intercambio de fichas.

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

   Si el comando devuelve un error `NoSuchEntity`, el rol no existe y debe crearlo. Para obtener más información acerca de cómo crear y configurar el rol, consulte Autorizar a los dispositivos principales a interactuar con AWS los servicios.

3. Compruebe que el rol tenga una relación de confianza que permita al proveedor de AWS IoT credenciales asumirlo. La respuesta del paso anterior contiene un `AssumeRolePolicyDocument`, que define las relaciones de confianza del rol. El rol debe definir una relación de confianza que permite a `credentials.iot.amazonaws.com` asumir el rol. Este documento debería verse similar al siguiente ejemplo.

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

   Si las relaciones de confianza del rol no permiten asumir `credentials.iot.amazonaws.com`, debe agregar esta relación de confianza al rol. Para obtener más información, consulte Modificación de un rol en la Guía del usuario de AWS Identity and Access Management .

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

Es posible que aparezca este error cuando un dispositivo principal no puede obtener AWS las credenciales del servicio de intercambio de fichas. El código de estado HTTP 403 indica que este error se produjo porque las AWS IoT políticas del dispositivo principal no conceden el `iot:AssumeRoleWithCertificate` permiso para el alias de AWS IoT rol del dispositivo principal.

Revisa las AWS IoT políticas del dispositivo principal y añade el `iot:AssumeRoleWithCertificate` permiso para el alias de AWS IoT rol del dispositivo principal. El mensaje de error incluye el alias de AWS IoT rol actual del dispositivo principal. Para obtener más información sobre este permiso y sobre cómo actualizar las AWS IoT políticas del dispositivo principal, consulte AWS IoT Política mínima para los dispositivos AWS IoT Greengrass V2 principales yActualice la AWS IoT política de un dispositivo principal.

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

Es posible que aparezca este error cuando el componente intente solicitar AWS credenciales y no pueda conectarse al servicio de intercambio de fichas.

Haga lo siguiente:

- Compruebe que el componente declara una dependencia del componente del servicio de intercambio de token, `aws.greengrass.TokenExchangeService`. Si no es así, agregue la dependencia y vuelva a implementar el componente.
- Si el componente se ejecuta en docker, asegúrese de aplicar la configuración de red y las variables de entorno correctas, según Utilice AWS las credenciales en los componentes del contenedor de Docker (Linux).
- Si el componente está escrito en Nodejs, defina dns. setDefaultResultOrdene para. **`ipv4first`**
- Compruebe si hay `/etc/hosts` para una entrada que comience por `::1` y contenga `localhost`. Elimine la entrada para ver si provocó que el componente se conectara al servicio de intercambio de token en una dirección incorrecta.

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

Es posible que aparezca este error cuando el componente no ejecuta el servicio de intercambio de fichas y un componente intenta solicitar AWS credenciales.

Haga lo siguiente:

- Compruebe que el componente declara una dependencia del componente del servicio de intercambio de token, `aws.greengrass.TokenExchangeService`. Si no es así, agregue la dependencia y vuelva a implementar el componente.
- Compruebe si el componente utiliza AWS credenciales `install` durante su ciclo de vida. AWS IoT Greengrass no garantiza la disponibilidad del servicio de intercambio de fichas durante el `install` ciclo de vida. Actualice el componente para trasladar el código que usa las credenciales de AWS en el ciclo de vida `startup` o `run`, a continuación, vuelva a implementar el componente.

copyFrom: <configurationPath> is already a container, not a leaf

Es posible que aparezca este error al cambiar un valor de configuración de un tipo de contenedor (una lista o un objeto) a un tipo que no es de contenedor (cadena, número o booleano). Haga lo siguiente:

1. Compruebe la receta del componente para ver si su configuración predeterminada establece ese valor de configuración en una lista o un objeto. Si es así, elimine o cambie ese valor de configuración.
2. Cree una implementación para restablecer ese valor de configuración a su valor predeterminado. Para obtener más información, consulte Crear implementaciones y Actualización de las configuraciones de los componentes.

A continuación, puede establecer ese valor de configuración en una cadena, un número o un booleano.

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'

Es posible que aparezca este error en los registros del núcleo de Greengrass cuando el componente administrador de aplicaciones de Docker intenta descargar una imagen de Docker de un repositorio privado de Amazon Elastic Container Registry (Amazon ECR). Este error se produce si utiliza el ayudante de credenciales de Docker (`docker-credential-wincred`) `wincred`. Como resultado, Amazon ECR no puede almacenar las credenciales de inicio de sesión.

Realice una de las siguientes acciones:

- Si no utiliza el ayudante de credenciales de Docker `wincred`, elimine el programa `docker-credential-wincred` del dispositivo principal.
- Si utiliza el ayudante de credenciales de Docker `wincred`, haga lo siguiente:
  1. Cambie el nombre del programa `docker-credential-wincred` en el dispositivo principal. Sustituya `wincred` por un nombre nuevo para el ayudante de credenciales de Docker de Windows. Por ejemplo, puede cambiarle el nombre a `docker-credential-wincredreal`.
  2. Actualice la opción `credsStore` en el archivo de configuración de Docker (`.docker/config.json`) para usar el nuevo nombre del ayudante de credenciales de Docker de Windows. Por ejemplo, si ha cambiado el nombre del programa a `docker-credential-wincredreal`, actualice la opción `credsStore` a `wincredreal`.

     ```
     {
       "credsStore": "wincredreal"
     }
     ```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

Es posible que aparezca este error en un dispositivo principal de Windows cuando el usuario del sistema que ejecuta los procesos del componente, por ejemplo `ggc_user`, tiene una contraseña caducada. Como resultado, el software AWS IoT Greengrass Core no puede ejecutar los procesos de los componentes como ese usuario del sistema.

Cómo actualizar la contraseña de un usuario del sistema de Greengrass

1. Ejecute el siguiente comando como administrador para establecer la contraseña del usuario. *ggc_user*Sustitúyalo por el usuario del sistema y *password* sustitúyalo por la contraseña que desee configurar.

   ```
   net user ggc_user password
   ```

2. Utilice la PsExec utilidad para almacenar la nueva contraseña del usuario en la instancia de Credential Manager de la LocalSystem cuenta. *password*Sustitúyala por la contraseña de usuario que hayas establecido.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

> **Tip**
>
> Según su configuración de Windows, es posible que la contraseña del usuario caduque en una fecha futura. Para garantizar que sus aplicaciones de Greengrass sigan funcionando, controle cuándo caduca la contraseña y actualícela antes de que caduque. También puede configurar la contraseña del usuario para que nunca caduque.
>
> - Para comprobar cuándo caducan un usuario y su contraseña, ejecute el siguiente comando.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para configurar la contraseña de un usuario para que no caduque nunca, ejecute el siguiente comando.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Si utilizas Windows 10 o una versión posterior, donde el wmiccomando está en desuso, ejecuta el siguiente PowerShell comando.
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Al actualizar la versión 2.0.7 del administrador de flujos a una versión entre la versión 2.0.8 y la versión 2.0.11, es posible que aparezca el siguiente error en los registros del componente administrador de flujos si el componente no se inicia.

```
2021-07-16T00:54:58.568Z [INFO] (Copier) aws.greengrass.StreamManager:
 stdout. Caused by: com.fasterxml.jackson.databind.JsonMappingException:
 Instant exceeds minimum or maximum instant (through reference chain:
 com.amazonaws.iot.greengrass.streammanager.export.PersistedSuccessExportStatesV1["lastExportTi
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
2021-07-16T00:54:58.579Z [INFO] (Copier) aws.greengrass.StreamManager: stdout.
 Caused by: java.time.DateTimeException: Instant exceeds minimum or maximum instant.
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
```

Si implementó el administrador de flujos versión 2.0.7 y desea actualizar a una versión posterior, debe actualizar al administrador de flujos versión 2.0.12 directamente. Para obtener más información sobre el componente administrador de flujos, consulte Administrador de flujos.

Problemas con los componentes de la función de Lambda del dispositivo principal

Solución de problemas con los componentes de la función de Lambda en los dispositivos principales.

Temas

- The following cgroup subsystems are not mounted: devices, memory
- ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

The following cgroup subsystems are not mounted: devices, memory

Es posible que vea este error al ejecutar una función de Lambda en contenedores en los siguientes casos:

- El dispositivo principal no tiene cgroup v1 activado para la memoria o los cgroups del dispositivo.

- El dispositivo principal tiene habilitada la versión 2 de cgroups. Las funciones de Lambda de Greengrass requieren cgroups v1 y los cgroups v1 y v2 se excluyen mutuamente.

Para habilitar cgroups v1, inicie el dispositivo con los siguientes parámetros del kernel de Linux.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

> **Tip**
>
> En una Raspberry Pi, edite el archivo `/boot/cmdline.txt` para configurar los parámetros del núcleo del dispositivo.

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

Es posible que aparezca este error al ejecutar una función Lambda de la versión 1, que utiliza el SDK AWS IoT Greengrass principal, en un dispositivo principal de la versión 2 sin especificar una suscripción en el componente del router de suscripciones anterior. Para solucionar este problema, implemente y configure el enrutador de suscripciones antiguo para especificar las suscripciones necesarias. Para obtener más información, consulte Cómo importar una función de Lambda V1.

Versión del componente descontinuada

Es posible que vea una notificación en su panel de estado personal (PHD) cuando se deje de fabricar una versión de un componente de su dispositivo principal. La versión del componente envía esta notificación a su PHD en un plazo de 60 minutos desde su descontinuación.

Para ver qué implementaciones necesita revisar, haga lo siguiente mediante la AWS Command Line Interface:

1. Ejecute el siguiente comando para obtener una lista de sus dispositivos principales.

   ```
   aws greengrassv2 list-core-devices
   ```

2. Ejecute el siguiente comando para recuperar el estado de los componentes en cada dispositivo principal del Paso 1. Reemplace *coreDeviceName* con el nombre de cada dispositivo principal para consultar.

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

3. Reúna los dispositivos principales con la versión de componentes discontinuada instalada en los pasos anteriores.
4. Ejecute el siguiente comando para recuperar el estado de todos los trabajos de implementación para un dispositivo principal del Paso 3. Reemplace *coreDeviceName* por el nombre del dispositivo principal que se va a consultar.

   ```
   aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
   ```

   La respuesta contiene la lista de trabajos de implementación para el dispositivo principal. Puede revisar la implementación para elegir otra versión del componente. Para obtener información sobre cómo revisar las implementaciones, consulte Revisar las implementaciones.

Problemas con la interfaz de la línea de comandos de Greengrass

Solucionar problemas con la CLI de Greengrass.

Temas

- java.lang.RuntimeException: Unable to create ipc client

java.lang.RuntimeException: Unable to create ipc client

Es posible que vea este error al ejecutar un comando de la CLI de Greengrass y especificar una carpeta raíz diferente a la que está instalado el software AWS IoT Greengrass principal.

Realice una de las siguientes acciones para establecer la ruta raíz y */greengrass/v2* sustitúyala por la ruta a la instalación del software AWS IoT Greengrass principal:

- Establezca la variable de entorno `GGC_ROOT_PATH` en */greengrass/v2*.
- Agregue los argumentos `--ggcRootPath` */greengrass/v2* a su comando como se muestra en el siguiente ejemplo.

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface problemas

Solucionar AWS CLI problemas para. AWS IoT Greengrass V2

Temas

- Error: Invalid choice: 'greengrassv2'

Error: Invalid choice: 'greengrassv2'

Es posible que aparezca este error al ejecutar un AWS IoT Greengrass V2 comando con AWS CLI (por ejemplo,`aws greengrassv2 list-core-devices`).

Este error indica que tienes una versión de la AWS CLI que no es compatible AWS IoT Greengrass V2. Para AWS IoT Greengrass V2 utilizarla con AWS CLI, debe tener una de las siguientes versiones o una posterior:

- Versión AWS CLI V1 mínima: v1.18.197
- Versión AWS CLI V2 mínima: v2.1.11

> **Tip**
>
> Puede ejecutar el siguiente comando para comprobar la versión de la AWS CLI que dispone.
>
> ```
> aws --version
> ```

Para resolver este problema, actualícelo AWS CLI a una versión posterior que sea compatible AWS IoT Greengrass V2. Para obtener más información, consulte Instalar, actualizar y desinstalar la AWS CLI en la Guía del usuario de AWS Command Line Interface .

Códigos de error de implementación detallados

Utilice los códigos de error y las soluciones de estas secciones para ayudar a resolver problemas con la implementación de componentes al utilizar el núcleo de Greengrass versión 2.8.0 o posterior.

El núcleo de Greengrass informa los errores de implementación como una jerarquía desde el código menos específico hasta el más específico disponible. Puede utilizar esta jerarquía para determinar el motivo de un error de implementación. Por ejemplo, la siguiente es una posible jerarquía de errores:

- DEPLOYMENT_FAILURE
  - ARTIFACT_DOWNLOAD_ERROR
    - IO_ERROR
      - DISK_SPACE_CRITICAL

Los códigos de error se organizan en tipos. Cada tipo representa una clase de errores que pueden producirse. AWS IoT Greengrass informa de estos tipos de errores en la consola, la API y AWS CLI. Puede haber más de un tipo de error, según los errores informados en la jerarquía de errores. En el ejemplo anterior, el tipo de error devuelto es `DEVICE_ERROR`.

Los tipos son:

- PERMISSION_ERROR: se denegó el acceso a una operación que requiere permiso.
- REQUEST_ERROR: se ha producido un error debido a un problema en el documento de implementación.
- COMPONENT_RECIPE_ERROR: se ha producido un error debido a un problema en la receta de un componente.
- AWS_COMPONENT_ERROR: se produjo un error al iniciar o eliminar un componente AWS proporcionado.
- USER_COMPONENT_ERROR: se produjo un error al iniciar o eliminar un componente de usuario.
- COMPONENT_ERROR: se produjo un error al iniciar o eliminar un componente, pero el núcleo de Greengrass no pudo determinar si el componente es un componente proporcionado por AWS o un componente de usuario.
- DEVICE_ERROR — Se ha producido un error en el dispositivo local I/O o en otro dispositivo.
- DEPENDENCY_ERROR: una implementación no pudo descargar un artefacto de Amazon S3 ni extraer una imagen de un registro de ECR.
- HTTP_ERROR: se ha producido un error con una solicitud HTTP.
- NETWORK_ERROR: se ha producido un error en la red del dispositivo.
- NUCLEUS_ERROR: el núcleo de Greengrass no pudo localizar un componente o no pudo encontrar la versión del núcleo activo.

- SERVER_ERROR: un servidor devolvió un error 500 en respuesta a una solicitud.
- CLOUD_SERVICE_ERROR: se ha producido un error en el servicio en la nube de AWS IoT Greengrass .
- UNKNOWN_ERROR: el componente lanzó una excepción no comprobada.

Muchos de los errores de esta sección contienen información adicional en los AWS IoT Greengrass registros principales. Estos registros se almacenan en el sistema de archivos local del dispositivo principal. Hay registros para el software AWS IoT Greengrass principal y para cada componente individual. Para obtener información sobre cómo acceder a los registros, consulte Acceso a los registros del sistema de archivos.

Error de permiso

ACCESS_DENIED

Es posible que aparezca este error cuando una operación de AWS servicio devuelva un error 403 porque los permisos no están configurados correctamente. Consulte el código de error más específico para obtener más información.

GET_DEPLOYMENT_CONFIGURATION_ACCESS_DENIED

Es posible que aparezca este error cuando la AWS IoT política no permita llamar a la `GetDeploymentConfiguration` operación. Agregue el permiso `greengrass::GetDeploymentConfiguration` a la política del dispositivo principal.

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

Es posible que aparezca este error cuando la AWS IoT política principal de dispositivos no permita el `greengrass:GetComponentVersionArtifact` permiso. Agregue el permiso a la política del dispositivo principal.

RESOLVE_COMPONENT_CANDIDATES_ACCESS_DENIED

Es posible que aparezca este error si la AWS IoT política de dispositivos principales no permite el `greengrass:ResolveComponentCandidates` permiso. Agregue el permiso a la política del dispositivo principal.

GET_ECR_CREDENTIAL_ERROR

Es posible que aparezca este error cuando la implementación no se pueda autenticar con un registro privado en ECR. Compruebe el registro para ver si hay un error específico y, a continuación, vuelva a intentar la implementación.

USER_NOT_AUTHORIZED_FOR_DOCKER

Es posible que aparezca este error cuando el usuario de Greengrass no esté autorizado a usar Docker. Asegúrese de ejecutar Greengrass como raíz o de que el usuario esté agregado al grupo `docker`. A continuación, intente la implementación de nuevo.

S3_ACCESS_DENIED

Es posible que aparezca este error cuando una operación de Amazon S3 devuelva un error 403. Compruebe los códigos o registros de error adicionales para obtener más información.

S3_HEAD_OBJECT_ACCESS_DENIED

Es posible que aparezca este error cuando la función de intercambio de token del dispositivo no permita que el software AWS IoT Greengrass Core descargue el artefacto componente desde la URL del objeto de S3 que especificó en la receta del componente o cuando el artefacto componente no está disponible. Compruebe que el rol de intercambio de token permita `s3:GetObject` de la URL del objeto de S3 en la que el artefacto está disponible y que el artefacto está presente.

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

Es posible que aparezca este error cuando el rol de intercambio de token del dispositivo no conceda el permiso `s3:GetBucketLocation` para el bucket de Amazon S3 en el que está disponible el artefacto. Compruebe que el dispositivo concede el permiso y, a continuación, vuelva a intentar la implementación.

S3_GET_OBJECT_ACCESS_DENIED

Es posible que aparezca este error cuando la función de intercambio de token del dispositivo no permita que el software AWS IoT Greengrass Core descargue el artefacto componente desde la URL del objeto de S3 que especificó en la receta del componente o cuando el artefacto componente no está disponible. Compruebe que el rol de intercambio de token permita `s3:GetObject` de la URL del objeto de S3 en la que el artefacto está disponible y que el artefacto está presente.

Error de solicitud

NUCLEUS_MISSING_REQUIRED_CAPABILITIES

Es posible que reciba este error cuando la versión del núcleo en la implementación no puede realizar una operación solicitada, como descargar una configuración grande o establecer límites

de recursos de Linux. Vuelva a intentar la implementación con una versión de núcleo que sea compatible con la operación.

MULTIPLE_NUCLEUS_RESOLVED_ERROR

Es posible que aparezca este error cuando una implementación intenta implementar varios componentes del núcleo. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

COMPONENT_CIRCULAR_DEPENDENCY_ERROR

Es posible que aparezca este error cuando dos componentes de la implementación dependen uno del otro. Revise la configuración de los componentes para que los componentes de la implementación no dependan unos de otros.

UNAUTHORIZED_NUCLEUS_MINOR_VERSION_UPDATE

Es posible que aparezca este error cuando un componente de su implementación requiera una actualización de la versión secundaria del núcleo, pero esa versión no esté especificada en la implementación. Esto ayuda a reducir las actualizaciones accidentales de versiones secundarias para los componentes que dependen de una versión diferente. Incluya la nueva versión secundaria del núcleo en la implementación.

MISSING_DOCKER_APPLICATION_MANAGER

Es posible que aparezca este error al implementar un componente de Docker sin implementar el administrador de aplicaciones de Docker. Asegúrese de que su implementación incluya el administrador de aplicaciones de Docker.

MISSING_TOKEN_EXCHANGE_SERVICE

Es posible que aparezca este error cuando la implementación quiera descargar un artefacto de imagen de Docker de un registro ECR privado sin implementar el servicio de intercambio de token. Asegúrese de que su implementación incluya el servicio de intercambio de token.

COMPONENT_VERSION_REQUIREMENTS_NOT_MET

Es posible que aparezca este error cuando haya un conflicto de restricciones de versión o no exista una versión de un componente. Para obtener más información, consulte Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>.

THROTTLING_ERROR

Es posible que aparezca este error cuando una operación AWS de servicio supere una cuota tarifaria. Reintente la implementación.

CONFLICTED_REQUEST

Este error puede aparecer cuando una operación de AWS servicio devuelve un error 409 porque la implementación intenta realizar más de una operación a la vez. Reintente la implementación.

RESOURCE_NOT_FOUND

Es posible que aparezca este error cuando una operación de AWS servicio devuelva un error 404 porque no se pudo encontrar un recurso. Compruebe el recurso que falta en el registro.

RUN_WITH_CONFIG_NOT_VALID

Es posible que aparezca este error cuando la información `posixUser`, `posixGroup` o `windowsUser` especificada para ejecutar el componente no sea válida. Compruebe que el usuario es válido y, a continuación, vuelva a intentar la implementación.

UNSUPPORTED_REGION

Es posible que aparezca este error si la región especificada para la implementación no es compatible con AWS IoT Greengrass. Compruebe la región y vuelva a intentar la implementación.

IOT_CRED_ENDPOINT_NOT_VALID

Es posible que aparezca este error cuando el punto final de AWS IoT credenciales especificado en la configuración no sea válido. Compruebe el punto de conexión e intente realizar la solicitud de nuevo.

IOT_DATA_ENDPOINT_NOT_VALID

Es posible que aparezca este error cuando el punto final de AWS IoT datos especificado en la configuración no sea válido. Compruebe el punto de conexión e intente realizar la solicitud de nuevo.

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

Es posible que aparezca este error cuando el artefacto del componente no esté disponible en la URL del objeto de S3 que especificó en la receta del componente. Compruebe que ha subido el artefacto al bucket de S3 y que el URI del artefacto coincide con la URL del objeto de S3 del artefacto en el bucket.

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

Es posible que este error se produzca cuando no encuentre el bucket de Amazon S3. Compruebe que el bucket existe y vuelva a intentar la implementación.

S3_GET_OBJECT_RESOURCE_NOT_FOUND

Es posible que aparezca este error cuando el artefacto del componente no esté disponible en la URL del objeto de S3 que especificó en la receta del componente. Compruebe que ha subido el artefacto al bucket de S3 y que el URI del artefacto coincide con la URL del objeto de S3 del artefacto en el bucket.

IO_MAPPING_ERROR

Es posible que aparezca este error cuando se produce un I/O error al analizar el documento o la receta de implementación. Compruebe los códigos o registros de error adicionales para obtener más información.

Error en la receta del componente

RECIPE_PARSE_ERROR

Es posible que aparezca este error si no se ha podido analizar la receta de implementación porque hay un error en la estructura de la receta. Compruebe que la receta tiene el formato correcto y vuelva a intentar la implementación.

RECIPE_METADATA_PARSE_ERROR

Es posible que aparezca este error cuando no se hayan podido analizar los metadatos de la receta de implementación descargados de la nube. Contacto Soporte.

ARTIFACT_URI_NOT_VALID

Es posible que aparezca este error cuando el URI de un artefacto de una receta no tenga el formato correcto. Compruebe en el registro el URI que no es válido, actualice el URI en la receta y vuelva a intentar la implementación.

S3_ARTIFACT_URI_NOT_VALID

Es posible que aparezca este error cuando el URI de Amazon S3 de un artefacto en una receta no sea válido. Compruebe en el registro el URI que no es válido, actualice el URI en la receta y vuelva a intentar la implementación.

DOCKER_ARTIFACT_URI_NOT_VALID

Es posible que reciba este error cuando el URI de Docker de un artefacto en una receta no es válido. Compruebe en el registro el URI que no es válido, actualice el URI en la receta y vuelva a intentar la implementación.

EMPTY_ARTIFACT_URI

Es posible que reciba este error cuando el URI de un artefacto en una receta no está especificado. Compruebe el registro en busca del artefacto al que le falta un URI, actualice el URI en la receta y luego intente la implementación nuevamente.

EMPTY_ARTIFACT_SCHEME

Es posible que aparezca este error cuando no se haya definido un esquema de URI para un artefacto. Compruebe en el registro el URI que no es válido, actualice el URI en la receta y vuelva a intentar la implementación.

UNSUPPORTED_ARTIFACT_SCHEME

Es posible que aparezca este error cuando la versión de núcleo en ejecución no admite un esquema de URI. Un URI no es válido o necesita actualizar la versión de núcleo. Si el URI no es válido, verifique el registro en busca del URI que no es válido, actualice el URI en la receta y, luego, intente la implementación nuevamente.

RECIPE_MISSING_MANIFEST

Es posible que aparezca este error cuando la sección del manifiesto no esté incluida en la receta. Agregue el manifiesto a la receta y vuelva a intentar la implementación.

RECIPE_MISSING_ARTIFACT_HASH_ALGORITHM

Es posible que aparezca este error cuando se especifica un artefacto que no es local dentro de una receta sin un algoritmo hash. Agregue el algoritmo al artefacto y, a continuación, vuelva a realizar la solicitud.

ARTIFACT_CHECKSUM_MISMATCH

Es posible que aparezca este error cuando un artefacto descargado tenga un resumen diferente al especificado en la receta. Asegúrese de que la receta contiene el resumen correcto y, a continuación, vuelva a intentar la implementación. Para obtener más información, consulte Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption..

COMPONENT_DEPENDENCY_NOT_VALID

Es posible que aparezca este error cuando el tipo de dependencia especificado en una receta de implementación no sea válido. Compruebe la receta y vuelva a realizar la solicitud de nuevo.

CONFIG_INTERPOLATE_ERROR

Es posible que aparezca este error al interpolar una variable de receta. Compruebe el registro para obtener más detalles.

IO_MAPPING_ERROR

Es posible que aparezca este error cuando se produce un I/O error al analizar el documento o la receta de despliegue. Compruebe los códigos o registros de error adicionales para obtener más información.

AWS error de componente, error de componente de usuario, error de componente

Los siguientes códigos de error se devuelven cuando hay un problema con un componente. El tipo de error real informado depende del componente específico que generó el error. Si el núcleo de Greengrass identifica el componente como uno proporcionado por AWS IoT Greengrass, regresa. `AWS_COMPONENT_ERROR` Si el componente se identifica como un componente de usuario, el núcleo de Greengrass devuelve `USER_COMPONENT_ERROR`. Si el núcleo de Greengrass no puede identificarlo, devuelve `COMPONENT_ERROR`.

COMPONENT_UPDATE_ERROR

Es posible que aparezca este error cuando un componente no se actualice durante una implementación. Compruebe los códigos de error adicionales o consulte el registro para ver la causa del error.

COMPONENT_BROKEN

Es posible que aparezca este error cuando un componente se rompe durante una implementación. Compruebe el registro de componentes para ver los detalles del error y, a continuación, vuelva a intentar la implementación.

REMOVE_COMPONENT_ERROR

Es posible que aparezca este error cuando el núcleo no puede eliminar un componente durante una implementación. Compruebe el registro para ver los detalles del error y, a continuación, vuelva a intentar la implementación.

COMPONENT_BOOTSTRAP_TIMEOUT

Es posible que reciba este error cuando la tarea de arranque de un componente tarde más que el tiempo de espera configurado. Aumente el tiempo de espera o reduzca el tiempo de ejecución de la tarea de arranque y, a continuación, vuelva a intentar la implementación.

COMPONENT_BOOTSTRAP_ERROR

Es posible que aparezca este error cuando la tarea de arranque de un componente tiene un error. Revise el registro para ver los detalles del error y, luego, intente la implementación nuevamente.

COMPONENT_CONFIGURATION_NOT_VALID

Es posible que aparezca este error cuando el núcleo no pueda validar la configuración implementada para el componente. Revise el registro para ver los detalles del error y, luego, intente la implementación nuevamente.

Error del dispositivo

IO_WRITE_ERROR

Es posible que aparezca este error al escribir en un archivo. Compruebe el registro para obtener más detalles.

IO_READ_ERROR

Es posible que aparezca este error al leer un archivo. Compruebe el registro para obtener más detalles.

DISK_SPACE_CRITICAL

Es posible que aparezca este error cuando no hay suficiente espacio en el disco para completar una solicitud de implementación. Debe tener al menos 20 Mb de espacio disponible o suficiente para guardar un artefacto más grande. Libere espacio en el disco e intente realizar la implementación de nuevo.

IO_FILE_ATTRIBUTE_ERROR

Es posible que aparezca este error cuando el tamaño del archivo existente no se pueda recuperar del sistema de archivos. Compruebe el registro para obtener más detalles.

SET_PERMISSION_ERROR

Es posible que aparezca este error cuando no se puedan establecer los permisos en un artefacto o directorio de artefactos descargado. Compruebe el registro para obtener más detalles.

IO_UNZIP_ERROR

Es posible que aparezca este error cuando no se pueda descomprimir un artefacto. Compruebe el registro para obtener más detalles.

LOCAL_RECIPE_NOT_FOUND

Es posible que aparezca este error si no se encuentra la copia local del archivo de una receta. Vuelva a intentar la implementación.

LOCAL_RECIPE_CORRUPTED

Es posible que aparezca este error si la copia local de la receta ha cambiado desde que se descargó. Elimine la copia existente de la receta y vuelva a intentar la implementación.

LOCAL_RECIPE_METADATA_NOT_FOUND

Es posible que reciba este error cuando no se pueda encontrar la copia local del archivo de metadatos de la receta. Vuelva a intentar la implementación.

LAUNCH_DIRECTORY_CORRUPTED

Es posible que aparezca este error si el directorio utilizado para lanzar el núcleo de Greengrass (`/greengrass/v2/alts/current`) se ha modificado desde la última vez que se inició el núcleo. Reinicie el núcleo y, a continuación, vuelva a intentar la implementación.

HASHING_ALGORITHM_UNAVAILABLE

Es posible que aparezca este error si la distribución Java del dispositivo no admite el algoritmo de hash requerido o cuando el algoritmo de hash especificado en la receta de un componente no es válido.

DEVICE_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNLOAD

Es posible que aparezca este error cuando hay un error en la configuración del dispositivo que impide que la implementación descargue el artefacto de Amazon S3 o de la nube de Greengrass.

Compruebe el registro para ver si hay un error específico y, a continuación, vuelva a intentar la implementación.

Error de dependencia

DOCKER_ERROR

Es posible que aparezca este error al extraer una imagen de Docker. Compruebe los códigos o registros de error adicionales para obtener más información.

DOCKER_SERVICE_UNAVAILABLE

Es posible que aparezca este error si Greengrass no puede iniciar sesión en el registro de Docker. Compruebe el registro para ver si hay un error específico y, a continuación, vuelva a intentar la implementación.

DOCKER_LOGIN_ERROR

Es posible que aparezca este error cuando se produce un error inesperado al iniciar sesión en Docker. Compruebe el registro para ver si hay un error específico y, a continuación, vuelva a intentar la implementación.

DOCKER_PULL_ERROR

Es posible que aparezca este error cuando se produce un error inesperado al extraer una imagen de Docker del registro. Compruebe el registro para ver si hay un error específico y, a continuación, vuelva a intentar la implementación.

DOCKER_IMAGE_NOT_VALID

Es posible que aparezca este error cuando la imagen de Docker solicitada no existe. Compruebe el registro para ver si hay un error específico y, a continuación, vuelva a intentar la implementación.

DOCKER_IMAGE_QUERY_ERROR

Es posible que aparezca este error cuando se produzca un error inesperado al consultar Docker para ver las imágenes disponibles. Compruebe el registro para ver si hay un error específico y vuelva a intentar la implementación.

S3_ERROR

Es posible que aparezca este error al descargar un artefacto de Amazon S3. Compruebe los códigos o registros de error adicionales para obtener más información.

S3_RESOURCE_NOT_FOUND

Es posible que aparezca este error cuando una operación de Amazon S3 devuelva un error 404. Compruebe los códigos o registros de error adicionales para obtener más información.

S3_BAD_REQUEST

Es posible que aparezca este error cuando una operación de Amazon S3 devuelva un error 400. Compruebe el registro para ver si hay un error específico y vuelva a intentar la solicitud.

Error de HTTP

HTTP_REQUEST_ERROR

Es posible que aparezca este error cuando se produce un error al realizar una solicitud HTTP. Compruebe el registro para ver si hay un error específico.

DOWNLOAD_DEPLOYMENT_DOCUMENT_ERROR

Es posible que aparezca este error cuando se produce un error HTTP al descargar el documento de implementación. Compruebe el registro para ver si hay un error HTTP específico.

GET_GREENGRASS_ARTIFACT_SIZE_ERROR

Es posible que aparezca este error cuando se produce un error HTTP al obtener el tamaño de un artefacto de un componente público. Compruebe el registro para ver si hay un error HTTP específico.

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

Es posible que aparezca este error cuando se produce un error HTTP al descargar un artefacto de componente público. Compruebe el registro para ver si hay un error HTTP específico.

Error de red

NETWORK_ERROR

Es posible que aparezca este error cuando hay un problema de conexión durante una implementación. Compruebe la conexión del dispositivo a Internet e intente la implementación de nuevo.

Error de núcleo

BAD_REQUEST

Es posible que aparezca este error cuando una operación AWS en la nube devuelva un error 400. Consulte el registro para ver qué API provocó el error y, a continuación, consulte la página de actualización del software de Nucleus para comprobar si el problema se ha corregido en una versión posterior del núcleo o póngase en contacto con nosotros Soporte.

NUCLEUS_VERSION_NOT_FOUND

Es posible que aparezca este error cuando un dispositivo principal no pueda encontrar la versión del núcleo activo. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

NUCLEUS_RESTART_FAILURE

Es posible que aparezca este error cuando el núcleo no se reinicie durante una implementación que requiera un reinicio del núcleo. Consulte el registro del cargador para ver qué causó el error, luego consulte la página de actualización del software del núcleo para ver si el problema se ha corregido en una versión posterior del núcleo, o comuníquese con Soporte.

INSTALLED_COMPONENT_NOT_FOUND

Es posible que aparezca este error cuando el núcleo no pueda localizar un componente instalado. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

DEPLOYMENT_DOCUMENT_NOT_VALID

Es posible que aparezca este error cuando el dispositivo reciba un documento de implementación que no es válido. Compruebe los códigos de error adicionales o consulte el registro para ver la causa del error.

EMPTY_DEPLOYMENT_REQUEST

Es posible que aparezca este error cuando un dispositivo reciba una solicitud de implementación vacía. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

DEPLOYMENT_DOCUMENT_PARSE_ERROR

Es posible que aparezca este error cuando el formato de la solicitud de implementación no coincide con el formato esperado. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

Es posible que aparezca este error cuando la solicitud de implementación contiene metadatos de componentes que no son válidos. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

LAUNCH_DIRECTORY_CORRUPTED

Es posible que aparezca este error cuando mueva un dispositivo de Greengrass de un grupo de objetos a otro y luego de vuelta al grupo original, con implementaciones que requieren que Greengrass se reinicie. Para resolver el error, cree de nuevo el directorio de inicio de Greengrass en el dispositivo.

Para obtener más información, consulte Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service.

Error del servidor

SERVER_ERROR

Es posible que aparezca este error cuando una operación de AWS servicio devuelva un error 500 porque el servicio no puede procesar la solicitud en este momento. Vuelva a intentar la implementación más tarde.

S3_SERVER_ERROR

Es posible que aparezca este error cuando una operación de Amazon S3 devuelve un error 500. Compruebe los códigos o registros de error adicionales para obtener más información.

Error del servicio en la nube

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

Es posible que aparezca este error cuando el servicio en la nube de Greengrass envía una respuesta incompatible a la operación `ResolveComponentCandidates`. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

DEPLOYMENT_DOCUMENT_SIZE_EXCEEDED

Es posible que aparezca este error cuando el documento de implementación solicitado supere la cuota de tamaño máxima. Reduzca el tamaño del documento de implementación e intente realizar la implementación de nuevo.

GREENGRASS_ARTIFACT_SIZE_NOT_FOUND

Es posible que aparezca este error cuando Greengrass no pueda obtener el tamaño de un artefacto de componente público. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

DEPLOYMENT_DOCUMENT_NOT_VALID

Es posible que aparezca este error cuando el dispositivo reciba un documento de implementación que no es válido. Compruebe los códigos de error adicionales o consulte el registro para ver la causa del error.

EMPTY_DEPLOYMENT_REQUEST

Es posible que aparezca este error cuando un dispositivo reciba una solicitud de implementación vacía. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

DEPLOYMENT_DOCUMENT_PARSE_ERROR

Es posible que aparezca este error cuando el formato de la solicitud de implementación no coincide con el formato esperado. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

Es posible que aparezca este error cuando la solicitud de implementación contiene metadatos de componentes que no son válidos. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

Errores genéricos

Estos errores genéricos no tienen un tipo de error asociado.

DEPLOYMENT_INTERRUPTED

Es posible que aparezca este error cuando no se pueda completar una implementación debido a un cierre del núcleo u otro suceso externo. Compruebe los códigos o registros de error adicionales para obtener más información.

ARTIFACT_DOWNLOAD_ERROR

Es posible que aparezca este error cuando haya un problema al descargar un artefacto. Compruebe los códigos o registros de error adicionales para obtener más información.

NO_AVAILABLE_COMPONENT_VERSION

Es posible que aparezca este error cuando la versión de un componente no existe en la nube o de forma local, o si hay un conflicto de resolución de dependencias. Compruebe los códigos o registros de error adicionales para obtener más información.

COMPONENT_PACKAGE_LOADING_ERROR

Es posible que reciba este error cuando se produce un error al procesar los artefactos descargados. Compruebe los códigos o registros de error adicionales para obtener más información.

CLOUD_ API_ERROR

Es posible que aparezca este error cuando se produce un error al llamar a una API AWS de servicio. Compruebe los códigos o registros de error adicionales para obtener más información.

IO_ERROR

Es posible que aparezca este error cuando se produce un I/O error durante una implementación. Compruebe los códigos o registros de error adicionales para obtener más información.

COMPONENT_UPDATE_ERROR

Es posible que aparezca este error cuando un componente no se actualice durante una implementación. Compruebe los códigos de error adicionales o consulte el registro para ver la causa del error.

Error desconocido

DEPLOYMENT_FAILURE

Es posible que aparezca este error cuando se produce un error en una implementación porque se ha producido una excepción no comprobada. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

DEPLOYMENT_TYPE_NOT_VALID

Es posible que aparezca este error cuando el tipo de implementación no sea válido. Consulte el registro para ver la causa del error y, a continuación, consulte la página de actualización del software de núcleo para comprobar si el problema se ha corregido en una versión posterior o póngase en contacto con Soporte.

Códigos de estado de componentes detallados

Utilice los códigos de estado y las soluciones de estas secciones para resolver problemas con componentes al utilizar el núcleo de Greengrass versión 2.8.0 o posterior.

Muchos de los estados de este tema contienen información adicional en los registros de AWS IoT Greengrass Core. Estos registros se almacenan en el sistema de archivos local del dispositivo principal. Hay registros para cada componente individual. Para obtener información sobre cómo acceder a los registros, consulte Acceso a los registros del sistema de archivos.

INSTALL_ERROR

Esto puede aparecer cuando se produce un error al ejecutar un script de instalación. El código de error aparece en el registro de componentes. Compruebe si hay errores en el script de instalación y vuelva a implementar el componente.

INSTALL_CONFIG_NOT_VALID

Es posible que aparezca este error cuando no se haya podido completar la instalación de un componente porque la sección `install` de la receta no es válida. Consulte la sección de instalación de su receta para ver si hay errores y vuelva a intentar la implementación.

INSTALL_IO_ERROR

Esto puede aparecer cuando se produce un error de I/O durante la instalación de un componente. Consulte el registro de errores del componente para obtener más detalles sobre el error.

INSTALL_MISSING_DEFAULT_RUNWITH

Este error puede aparecer cuando AWS IoT Greengrass no puede determinar el usuario o el grupo que se va a utilizar al ejecutar un componente. Asegúrese de que la sección `runWith` de la receta de instalación incluya un usuario o grupo válido.

INSTALL_TIMEOUT

Este error puede aparecer si el script de instalación no ha finalizado dentro del tiempo de espera configurado. Aumente el periodo `Timeout` especificado en la sección `install` de la receta o modifique su script de instalación para que finalice dentro del tiempo de espera configurado.

STARTUP_ERROR

Esto puede aparecer cuando se produce un error al ejecutar un script de iniciación. El código de error aparece en el registro de componentes. Compruebe si hay errores en el script de instalación y vuelva a implementar el componente.

STARTUP_CONFIG_NOT_VALID

Es posible que aparezca este error cuando no se haya podido completar la instalación de un componente porque la sección `startup` de la receta no es válida. Consulte la sección de iniciación de su receta para ver si hay errores y vuelva a intentar la implementación.

STARTUP_IO_ERROR

Esto puede aparecer cuando se produce un error de I/O durante la iniciación de un componente. Consulte el registro de errores del componente para obtener más detalles sobre el error.

STARTUP_MISSING_DEFAULT_RUNWITH

Es posible que aparezca este error cuando AWS IoT Greengrass no pueda determinar el usuario o el grupo que se va a utilizar al ejecutar un componente. Asegúrese de que la sección `runWith` de la receta de iniciación incluya un usuario o grupo válido.

STARTUP_TIMEOUT

Este error puede aparecer si el script de iniciación no ha finalizado dentro del tiempo de espera configurado. Aumente el periodo `Timeout` especificado en la sección `startup` de la receta o modifique el script de iniciación para que finalice dentro del tiempo de espera configurado.

RUN_ERROR

Esto puede aparecer cuando se produce un error al ejecutar un script de componente. El código de error aparece en el registro de componentes. Compruebe si hay errores en el script y vuelva a implementar el componente.

RUN_MISSING_DEFAULT_RUNWITH

Es posible que aparezca este error cuando AWS IoT Greengrass no pueda determinar el usuario o el grupo que se va a utilizar al ejecutar un componente. Verifique que la sección `runWith` de su receta de ejecución incluya un usuario o grupo válido.

RUN_CONFIG_NOT_VALID

Es posible que aparezca este error cuando no se pudo ejecutar un componente porque la sección `run` de la receta no es válida. Consulte la sección de ejecución de su receta para ver si hay errores y vuelva a intentar la implementación.

RUN_IO_ERROR

Es posible que aparezca esto cuando se produce un error de I/O mientras el componente se está ejecutando. Consulte el registro de errores del componente para obtener más detalles sobre el error.

RUN_TIMEOUT

Es posible que reciba este error cuando el script de ejecución no finalizó dentro del tiempo de espera configurado. Aumente el periodo `Timeout` especificado en la sección `run` de la receta o modifique el script de ejecución para que finalice dentro del tiempo de espera configurado.

SHUTDOWN_ERROR

Esto puede aparecer cuando se produce un error al cerrar un script de componente. El código de error aparece en el registro de componentes. Compruebe el script de apagado para detectar errores e implemente su componente nuevamente.

SHUTDOWN_TIMEOUT

Es posible que reciba este error cuando el script de apagado no finalizó dentro del tiempo de espera configurado. Aumente el periodo `Timeout` especificado en la sección `shutdown`

de la receta o modifique el script de ejecución para que finalice dentro del tiempo de espera configurado.

Etiquete sus AWS IoT Greengrass Version 2 recursos

Las etiquetas le permiten organizar y administrar sus recursos en AWS IoT Greengrass. Puede utilizar etiquetas para asignar metadatos a los recursos y etiquetas en las políticas de IAM para definir el acceso condicional a sus recursos.

> **Note**
>
> Actualmente, las etiquetas de recursos de Greengrass no se admiten en los grupos de AWS IoT facturación ni en los informes de asignación de costes.

Uso de etiquetas en AWS IoT Greengrass V2

Puede usar etiquetas para clasificar sus AWS IoT Greengrass recursos por propósito, propietario, entorno o cualquier otra clasificación para su caso de uso. Cuando tiene muchos recursos del mismo tipo, las etiquetas lo ayudan a identificar más fácilmente un recurso específico.

Cada etiqueta está formada por una clave y un valor opcional, ambos definidos por el usuario. Por ejemplo, podría definir un conjunto de etiquetas para sus dispositivos principales que le permita realizar un seguimiento de ellos según los clientes que tienen los dispositivos. Le recomendamos que cree un conjunto de claves de etiqueta que cumpla sus necesidades para cada tipo de recurso. Si utiliza un conjunto coherente de claves de etiquetas, le será más fácil administrar los recursos.

Etiquete con Consola de administración de AWS

El editor de etiquetas del Consola de administración de AWS proporciona una forma centralizada y unificada de crear y administrar las etiquetas para los recursos de todos los AWS servicios. Para obtener más información, consulte Tag Editor en la Guía del usuario de Grupos de recursos de AWS .

Etiquete con la AWS IoT Greengrass V2 API

También puedes usar la AWS IoT Greengrass V2 API para trabajar con etiquetas. Antes de crear etiquetas, tenga en cuenta las restricciones de etiquetado. Para obtener más información, consulte este artículo sobre las convenciones de nomenclatura y el uso de etiquetas en la Referencia general de AWS.

- Para agregar etiquetas al crear un recurso, deberá definirlas en la propiedad `tags` del recurso.
- Para añadir etiquetas a un recurso existente o para actualizar los valores de las etiquetas, utilice la TagResourceoperación.
- Para eliminar etiquetas de un recurso, utilice la UntagResourceoperación.
- Para recuperar las etiquetas asociadas a un recurso, utilice la ListTagsForResourceoperación o describa el recurso e inspeccione su `tags` propiedad.

En la siguiente tabla, se enumeran los recursos que puedes etiquetar mediante la AWS IoT Greengrass V2 API y sus `Get` operaciones `Describe` y/u correspondientes`Create`.

Recursos etiquetables AWS IoT Greengrass V2

Recurso	Crear operación	Operación Describe o Get
Dispositivo principal	Ninguna. Ejecute el software AWS IoT Greengrass principal en un dispositivo para crear un dispositivo principal.	GetCoreDevice
Componente	CreateComponentVersion	DescribeComponent, GetComponent
Implementación	CreateDeployment	GetDeployment

Utilice las siguientes operaciones para ver y administrar etiquetas para los recursos que admiten etiquetas:

- TagResource— Añade etiquetas a un recurso o actualiza el valor de una etiqueta existente.
- ListTagsForResource— Muestra las etiquetas de un recurso.
- UntagResource— Elimina las etiquetas de un recurso.

Puede agregar o eliminar etiquetas de un recurso en cualquier momento. Para cambiar el valor de una clave de etiqueta, añada una etiqueta al recurso que defina la misma clave y el nuevo valor. El nuevo valor reemplaza al valor anterior. Puede establecer un valor como una cadena vacía, pero no puede definir un valor como nulo.

Al eliminar un recurso, también se eliminarán las etiquetas que este tenga asociadas.

Uso de etiquetas con políticas de IAM

En las políticas de IAM, puede utilizar etiquetas de recursos para controlar los permisos y el acceso de usuarios. Por ejemplo, las políticas pueden permitir a los usuarios crear solo aquellos recursos que tienen una etiqueta específica. Las políticas también puede limitar la creación o modificación de recursos que tengan determinadas etiquetas por parte de los usuarios.

> **Note**
>
> Si utiliza etiquetas para permitir o denegar el acceso de los usuarios a los recursos, debe denegar a los usuarios la capacidad de agregar o eliminar esas etiquetas para los mismos recursos. De lo contrario, un usuario podría eludir sus restricciones y obtener acceso a un recurso modificando sus etiquetas.

Puede utilizar los siguientes claves y valores de contexto de condición en el elemento `Condition` (también llamado bloque `Condition`) de una instrucción de política.

`greengrassv2:ResourceTag/`*`tag-key`*`: `*`tag-value`*

Permitir o denegar acciones en recursos con etiquetas específicas.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

Exigir que se utilice (o no se utilice) una etiqueta específica al crear o modificar un recurso etiquetable.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Exigir que se utilice (o no se utilice) un conjunto específico de claves de etiqueta al crear o modificar un recurso etiquetable.

> **Note**
>
> Las claves y valores de contexto de condición en una política de IAM se aplican solo a las acciones que tienen un recurso etiquetable como parámetro requerido. Por ejemplo, puede configurar el acceso condicional basado en etiquetas para GetComponent.

Para obtener más información, consulte Controlar el acceso a AWS los recursos mediante etiquetas de recursos y la referencia a la política JSON de IAM en la Guía del usuario de IAM.

Creación de AWS IoT Greengrass recursos con AWS CloudFormation

AWS IoT Greengrass está integrado con AWS CloudFormation un servicio que le ayuda a modelar y configurar sus AWS recursos para que pueda dedicar menos tiempo a crear y administrar sus recursos e infraestructura. Crea una plantilla que describe todos los AWS recursos que desea (como las versiones de los componentes y las implementaciones) y CloudFormation aprovisiona y configura esos recursos por usted.

Cuando la utilice CloudFormation, podrá reutilizar la plantilla para configurar los AWS IoT Greengrass recursos de forma coherente y repetida. Describa sus recursos una vez y, a continuación, aprovisione los mismos recursos una y otra vez en varias Cuentas de AWS regiones.

AWS IoT Greengrass y CloudFormation plantillas

Para aprovisionar y configurar recursos AWS IoT Greengrass y servicios relacionados, debe conocer CloudFormation las plantillas. Las plantillas son archivos de texto con formato JSON o YAML. Estas plantillas describen los recursos que desea aprovisionar en sus CloudFormation pilas. Si no estás familiarizado con JSON o YAML, puedes usar CloudFormation Designer para ayudarte a empezar con CloudFormation las plantillas. Para obtener más información, consulte ¿Qué es Designer de CloudFormation ? en la Guía del usuario de AWS CloudFormation .

AWS IoT Greengrass admite la creación de versiones e implementaciones de componentes en. CloudFormation Para obtener más información, incluyendo ejemplos de plantillas JSON y YAML para las versiones e implementaciones del componente, consulte la referencia del tipo de recurso de AWS IoT Greengrass en la Guía del usuario de AWS CloudFormation .

ComponentVersion ejemplo de plantilla

A continuación, se muestra la plantilla YAML para una versión de un componente simple. La receta de JSON incluye saltos de línea por motivos de legibilidad.

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
  TestSimpleComponentVersion:
```

```
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
      InlineRecipe: !Sub
        - "{\n
            \"RecipeFormatVersion\": \"2020-01-25\",\n
            \"ComponentName\": \"component1\",\n
            \"ComponentVersion\": \"${ComponentVersion}\",\n
            \"ComponentType\": \"aws.greengrass.generic\",\n
            \"ComponentDescription\": \"This\",\n
            \"ComponentPublisher\": \"You\",\n
            \"Manifests\": [\n
            {\n
              \"Platform\": {\n
              \"os\": \"darwin\"\n
            },\n
            \"Lifecycle\": {},\n
            \"Artifacts\": []\n
           },\n
           {\n
             \"Lifecycle\": {},\n
             \"Artifacts\": []\n
           }\n
          ],\n
          \"Lifecycle\": {\n
          \"install\": {\n
            \"script\": \"yuminstallpython\"\n
          }\n
         }\n
        }"
        - { ComponentVersion: !Ref ComponentVersion }
```

Ejemplo de plantilla de implementación

El siguiente es un archivo YAML que define una plantilla sencilla para una implementación.

```
Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
  TestDeployment:
    Type: AWS::GreengrassV2::Deployment
```

```
    Properties:
      Components:
        component1:
          ComponentVersion: !Ref ComponentVersion
      TargetArn: !Ref TargetArn
      DeploymentName: CloudFormationIntegrationTest
      DeploymentPolicies:
        FailureHandlingPolicy: DO_NOTHING
        ComponentUpdatePolicy:
          TimeoutInSeconds: 5000
          Action: SKIP_NOTIFY_COMPONENTS
        ConfigurationValidationPolicy:
          TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

Obtenga más información sobre CloudFormation

Para obtener más información CloudFormation, consulte los siguientes recursos:

- AWS CloudFormation
- AWS CloudFormation Guía del usuario
- CloudFormation Referencia de la API
- Guía del usuario de la interfaz de la línea de comandos de AWS CloudFormation

Software AWS IoT Greengrass Core de código abierto

El tiempo de ejecución de periferia AWS IoT Greengrass Version 2 (núcleo) y otros componentes del software AWS IoT Greengrass Core son de código abierto. Esto significa que puede revisar el código para solucionar problemas de interacción con sus aplicaciones. También puede personalizar y ampliar el software AWS IoT Greengrass Core para que se adapte a sus necesidades específicas de software y hardware.

Para obtener información sobre los repositorios de código abierto del software AWS IoT Greengrass Core, consulta la organización aws-greengrass en GitHub. El uso del software de código abierto se rige por la licencia de código abierto del repositorio de GitHub correspondiente.

El uso del software y los componentes de AWS IoT Greengrass Core que no estén sujetos a una licencia de código abierto se rige por la licencia de software principal de Greengrass.

Historial de documentos de la Guía para AWS IoT Greengrass V2 desarrolladores

En la siguiente tabla se describe la documentación de esta versión de AWS IoT Greengrass Version 2.

- Versión de la API: 30 de noviembre de 2020

Cambio	Descripción	Fecha
Publicada la versión 2.3.13 de Shadow Manager	La versión 2.3.13 de Shadow manager está disponibl e. Esta versión corrige un problema que provocaba que las actualizaciones ocultas locales no se sincroniz aran con la nube de forma intermitente, incluso cuando la configuración de la dirección de sincronización lo permitía.	18 de febrero de 2026
Lanzamiento del componente Greengrass nucleus lite v2.3.2	Está disponible la versión 2.3.2 del componente núcleo lite de Greengrass. Esta versión actualiza el archivo de versiones para informar correctamente sobre la versión de Nucleus.	12 de febrero de 2026
Lanzamiento del componente Greengrass nucleus lite v2.3.1	Está disponible la versión 2.3.1 del componente núcleo lite de Greengrass. Esta versión incluye numerosas correcciones de errores y mejoras.	11 de febrero de 2026

Lanzamiento de la versión 2.16.1 de Greengrass CLI	El componente CLI de Greengrass v2.16.1 está disponible.	23 de diciembre de 2025
AWS IoT Greengrass Actualización de software Core v2.16.1	Esta versión proporciona la versión 2.16.1 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	23 de diciembre de 2025
Publicada la versión 2.1.0 de System Log Forwarder	El reenviador de registros del sistema, versión 2.1.0, está disponible.	6 de noviembre de 2025
Lanzamiento de la versión 2.3.12 de Shadow Manager	Ya está disponible la versión 2.3.12 de Shadow manager.	6 de noviembre de 2025
Publicada la versión 2.3.11 de Log Manager	Está disponible el componente Log Manager v2.3.11.	6 de noviembre de 2025
Lanzamiento del componente Greengrass nucleus lite v2.3.0	Está disponible la versión 2.3.0 del componente núcleo lite de Greengrass. Esta versión añade compatibilidad con TPM 2.0, una gama más amplia de sistemas operativo s e IPC. RestartComponent También incluye diversas correcciones de errores y mejoras.	6 de noviembre de 2025
Lanzamiento de la versión 2.16.0 de Greengrass CLI	El componente CLI de Greengrass v2.16.0 está disponible.	6 de noviembre de 2025

AWS IoT Greengrass Actualización de software Core v2.16.0	Esta versión proporciona la versión 2.16.0 del componente núcleo de Greengrass y actualiza los componentes proporcionados. AWS	6 de noviembre de 2025
AWS IoT Greengrass Actualización de software Core v2.15.1	Esta versión proporciona la versión 2.15.1 del componente núcleo de Greengrass y actualiza los componentes proporcionados. AWS	6 de noviembre de 2025
Lanzamiento de la versión 2.0.1 del reenviador de registros del sistema	Ya está disponible la versión 2.0.1 del reenviador de registros del sistema. Esta versión actualiza la receta del componente para que sea compatible correctamente con los sistemas aarch64 (arm64).	2 de octubre de 2025
Lanzamiento del componente de la versión lite 2.2.2 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.2.2 del núcleo de Greengrass. Esta versión incluye numerosas correcciones de errores y mejoras.	18 de agosto de 2025
Lanzamiento de la versión 2.0.0 del reenviador de registros del sistema	Ya está disponible el componente de la versión 2.0.0 del reenviador de registros del sistema. Ya está disponible la versión inicial.	25 de julio de 2025

Lanzamiento del componente de la versión lite 2.2.1 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.2.1 del núcleo de Greengrass. Esta versión soluciona el problema que impedía que el núcleo obtuviera las credenciales de TES.	25 de julio de 2025
Lanzamiento del componente de la versión lite 2.2.0 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.2.0 del núcleo de Greengrass. Esta versión añade compatibilidad con el artefacto URIs de imagen del contenedor.	3 de julio de 2025
AWS IoT Greengrass Actualización de software Core v2.15.0	Esta versión proporciona la versión 2.15.0 del componente núcleo de Greengrass y actualiza los componentes proporcionados. AWS	3 de julio de 2025
Publicada la versión 1.1.3 de Secure Tunneling	Está disponible la versión 1.1.3 de Secure Tunneling. Esta versión actualiza el AWS IoT Device Client subyacente invocado por el componente de la versión 1.10.0 a la versión 1.10.1. También soluciona los problemas de compatibilidad con la biblioteca GNU C (glibc) en la versión 1.1.2 del componente de tunelización segura.	16 de mayo de 2025

Lanzamiento del componente de la versión lite 2.1.0 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.1.0 del núcleo de Greengrass. Esta versión agrega compatibilidad con el proxy HTTP. También incluye diversas correcciones de errores y mejoras.	5 de mayo de 2025
Lanzamiento de la versión 2.2.5 del administrador de secretos	Ya está disponible la versión 2.2.5 del administrador de secretos. Esta versión corrige un problema por el que un secreto no se recupera de la caché local Nube de AWS si no está presente en la memoria caché local.	16 de abril de 2025
AWS IoT Greengrass Actualización de software Core v2.14.3	Esta versión incluye la versión 2.14.3 del componente núcleo de Greengrass y actualiza los componentes proporcionados. AWS	11 de abril de 2025
Lanzamiento de la versión 2.2.4 del administrador de secretos	Ya está disponible la versión 2.2.4 del administrador de secretos. Esta versión reduce la frecuencia de las escrituras en el almacén de secretos local. El administrador de secretos ahora escribe en la tienda local solo cuando se actualizan los secretos.	8 de abril de 2025
Lanzamiento de la versión 2.14.3 de Greengrass CLI	El componente CLI de Greengrass v2.14.3 está disponible.	8 de abril de 2025

Publicada la versión 1.1.2 de Secure Tunneling	Está disponible la versión 1.1.2 de Secure Tunneling. Esta versión actualiza AWS IoT Device Client subyacent e invocado por el component e desde la versión 1.9.0 a la versión 1.10.0. La versión 1.1.2 de la tunelización segura también corrige el problema de transferencia de carga útil que impedía a los usuarios reenviar archivos de gran tamaño desde los dispositi vos principales de Greengras s V2 al dispositivo de origen mediante el túnel seguro.	27 de marzo de 2025
Lanzamiento de la versión 2.14.2 de Greengrass CLI	El componente CLI de Greengrass v2.14.2 está disponible.	27 de marzo de 2025
AWS IoT Greengrass Actualización de software Core v2.14.2	Esta versión proporciona la versión 2.14.2 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	27 de marzo de 2025

Lanzamiento de la versión 2.2.3 del administrador de secretos	Ya está disponible la versión 2.2.3 del administrador de secretos. Esta versión corrige un problema donde el administrador de secretos borra los secretos almacenados de forma local cuando el dispositivo principal está desconectado y el servicio de seguridad del dispositivo (como un HSM) no está disponible.	18 de marzo de 2025
Publicada la versión 2.2.1 de Stream Manager	Ya está disponible la versión 2.2.1 de Stream Manager. Esta versión soluciona un problema que impedía que el administrador de flujos exportara los mensajes a los destinos de Kinesis. Esta versión también mejora el rendimiento de las exportaciones del administrador de transmisión a destinos de Kinesis.	12 de marzo de 2025
Lanzamiento del componente de autenticación de dispositivos cliente, versión 2.5.3	El componente de autenticación de dispositivos cliente, versión 2.5.3, está disponible. Esta versión soluciona un problema que impedía que los dispositivos cliente se conectaran al dispositivo principal debido a que los certificados de cliente estaban desactualizados.	12 de marzo de 2025

AWS IoT Lanzamiento del complemento de aprovisionamiento de flotas v1.2.2	AWS IoT Está disponible la versión 1.2.2 del complemento de aprovisionamiento de flotas. Esta versión agrega compatibilidad con las rutas personalizadas para certificados de dispositivos principales (`certificatePath`) y claves privadas (`privateKeyPath`).	12 de marzo de 2025
Publicada la versión 1.1.1 de Secure Tunneling	Está disponible la versión 1.1.1 de Secure Tunneling. Esta versión agrega una configuración compatible con la versión lite del núcleo de Greengrass.	7 de febrero de 2025
Lanzamiento de la versión 2.14.1 de Greengrass CLI	El componente CLI de Greengrass, versión 2.14.1, está disponible.	7 de febrero de 2025
AWS IoT Greengrass Actualización de software Core v2.14.1	Esta versión proporciona la versión 2.14.1 del componente núcleo de Greengrass y actualiza los componentes proporcionados. AWS	7 de febrero de 2025
Lanzamiento del componente de la versión lite 2.0.2 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.0.2 del núcleo de Greengrass. Esta versión incluye correcciones de errores y mejoras generales.	6 de febrero de 2025

Lanzamiento del componente de la versión lite 2.0.1 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.0.1 del núcleo de Greengrass. Esta versión incluye correcciones de errores y mejoras generales.	28 de enero de 2025
Lanzamiento de la versión 2.14.0 de Greengrass CLI	El componente CLI de Greengrass v2.14.0 está disponible.	24 de diciembre de 2024
Lanzamiento de la versión 2.2.0 de Stream Manager	Ya está disponible la versión 2.2.0 de Stream Manager.	16 de diciembre de 2024
Lanzada la versión 2.3.10 de Shadow Manager	Ya está disponible la versión 2.3.10 de Shadow manager.	16 de diciembre de 2024
Publicada la versión 1.1.0 de Secure Tunneling	Está disponible la versión 1.1.0 de Secure Tunneling. Esta versión agrega compatibilidad de recetas para la versión lite del núcleo de Greengrass.	16 de diciembre de 2024
Nuevo componente emisor de telemetría de núcleo	Está disponible la versión 1.0.10 del componente emisor de telemetría del núcleo.	16 de diciembre de 2024
Lanzamiento del adaptador de protocolo Modbus-RTU, versión 2.1.10	Está disponible el componente adaptador de protocolo Modbus-RTU, versión 2.1.10.	16 de diciembre de 2024
Publicada la versión 2.3.9 del administrador de registros	Está disponible el componente Log Manager v2.3.9.	16 de diciembre de 2024

Lanzamiento de la consola de depuración local, versión 2.4.4	Está disponible el componente de la consola de depuración local, versión 2.4.4. Esta versión incluye correcciones de errores y mejoras generales.	16 de diciembre de 2024
Lanzada la versión 2.3.5 de Lambda Manager	Está disponible el componente Lambda Manager v2.3.5.	16 de diciembre de 2024
Publicada la versión 2.2.1 del detector de IP	Está disponible el componente detector IP v2.2.1.	16 de diciembre de 2024
Lanzamiento del componente de la versión lite 2.0.0 del núcleo de Greengrass	Ya está disponible el componente de la versión lite 2.0.0 del núcleo de Greengrass. Esta es la versión inicial.	16 de diciembre de 2024
Lanzada la versión 1.0.5 de Disk spooler	Está disponible el componente Disk Spooler v1.0.5.	16 de diciembre de 2024
Lanzamiento del componente de autenticación de dispositivos cliente, versión 2.5.2	El componente de autenticación de dispositivos cliente, versión 2.5.2, está disponible.	16 de diciembre de 2024

AWS IoT Greengrass Actualización de software Core v2.14.0	Esta versión proporciona la versión 2.14.0 del componente del núcleo de Greengrass y nuevas actualizaciones de la versión lite del núcleo de AWS IoT Greengrass . El AWS IoT Greengrass núcleo lite es un nuevo motor de ejecución , disponible para AWS IoT Greengrass la versión 2. Es una alternativa que ocupa menos memoria. Una buena opción para dispositivos con recursos limitados. Implementa un subconjunto de las funciones del núcleo con una mayor compatibilidad prevista para futuras versiones. Ya está disponible el código origen en GitHub. Con el tiempo de ejecución de la versión lite del núcleo de Greengrass, se puede realizar lo siguiente: • Implementar componentes en los dispositivos principales de Greengrass. Utilizar el mismo formato de receta, aunque algunas características avanzadas todavía pueden que no estén disponibles. • Las aplicaciones implementadas como componentes de Greengrass pueden	16 de diciembre de 2024

usar el dispositivo SDKs para acceder al IPC de Greengrass compatible APIs, como el acceso AWS IoT Core MQTT, el pub/sub local y el acceso a la configuración de Greengras s. Consulte la tabla de compatibilidad para ver la lista de IPC compatibles. APIs

- Algunos componentes AWS gestionados se han actualizado para que sean compatibles con nucleus lite. Consulte los componentes proporcionados por AWS para obtener una lista de los componentes compatibles.

Nuevas características:

- Utiliza menos memoria y espacio en disco (menos de 5 MB de RAM y menos de 5 MB de almacenamiento).
- Los componentes se integran con el administrador de servicios del sistema host (systemd para las plataformas Linux compatibles).

Aspectos a tener en cuenta:

- AWS IoT Greengrass Las recetas de nucleus lite distinguen mayúsculas de minúsculas. Verifique que la carcasa (llaves) correcta se utiliza como en la referencia de la receta https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html.
- El tiempo de ejecución de la versión lite del núcleo admite implementaciones del grupo de objetos, pero aún no admite el tipo de destino de implementación (único) del dispositivo principal. Para realizar la implementación en un único dispositivo de Greengrass, utilice un grupo de objetos que contenga solo ese dispositivo.
- El tiempo de ejecución de la versión lite del núcleo utiliza recursos de memoria limitados. La funcionalidad se adapta según el uso en la versión clásica del tiempo de ejecución, la cual puede fallar debido a que se superan los recursos disponibles en la versión lite. Esto incluye una limitación actual de

un máximo de 50 suscripciones a MQTT a la vez y límites máximos de tamaño e implementación de los archivos de recetas. Algunos de estos límites se pueden configurar en tiempo de compilación si usted compila el tiempo de ejecución de la versión lite.

- El tiempo de ejecución de la versión lite del núcleo no se incluye con Java. Para usar componentes que requieran Java, el sistema necesitará tener Java ya instalado, o bien se podrá utilizar un componente para instalar Java.
- Recomendamos compilar el tiempo de ejecución de la versión lite del núcleo desde la fuente y utilizar una compilación propia adaptada a su sistema. En el caso de los sistemas Yocto, ya está disponible una capa para integrar el tiempo de ejecución de la versión lite del núcleo en la imagen del sistema.
- Por ahora, la versión lite del núcleo asume que un sistema Linux utiliza systemd o que una imagen

de contenedor utiliza systemd.

- Si bien puede administrar los contenedores de Docker con scripts de recetas, aún no están disponibles los artefactos de contenedo res administrados por Greengrass.
- El motor de ejecución de nucleus lite aún no admite las claves almacenadas en un PKCS11 módulo. Si su caso de uso requiere que las claves se almacenen en un elemento seguro, la versión clásica del tiempo de ejecución clásico podrá admitirlas. Para evitar que se pierdan las credenciales de su dispositivo, asegúrese de que los dispositivos de producción utilicen un cifrado de disco completo.

Además, lanzaremos la versión 2.14.0 del núcleo junto con la introducción de la versión lite del núcleo. Esta actualización aporta mejoras significativas al núcleo actual de Greengrass.

Características y mejoras claves:

	• El nuevo soporte para terminales de doble pila permite la comunicación en IPv6 red. • Resiliencia mejorada contra los errores de reinicio del núcleo y la corrupción de directorios. • Se corrigieron las fugas de memoria en los cierres de PubSub suscripciones a IPC.	
Versión 2.1.13 del administrador de flujos publicada	Ya está disponible la versión 2.1.13 del administrador de flujos. Esta versión añade compatibilidad con el terminal FIPS para AWS IoT SiteWise	26 de agosto de 2024
Versión 2.3.9 del administrador de sombras publicada	Ya está disponible la versión 2.3.9 del administrador de sombras.	26 de agosto de 2024
Versión 2.1.9 del adaptador de protocolo Modbus-RTU publicada	Ya está disponible la versión 2.1.9 del componente adaptador de protocolo Modbus-RTU.	26 de agosto de 2024
Versión 2.3.8 del administrador de registros publicada	Ya está disponible la versión 2.3.8 del componente del administrador de registros.	26 de agosto de 2024

Versión 2.4.3 de la consola de depuración local publicada	Ya está disponible la versión 2.4.3 del componente de la consola de depuración local. Esta versión incluye correcciones de errores y mejoras generales.	26 de agosto de 2024
Versión 1.0.4 del spooler de disco publicada	Ya está disponible la versión 1.0.4 del spooler de disco.	26 de agosto de 2024
AWS IoT Greengrass Actualización de software Core v2.13.0	Este lanzamiento proporciona la versión 2.13.0 del componente de núcleo de Greengrass.	26 de agosto de 2024
Nuevo componente emisor de telemetría de núcleo	Ya está disponible la versión 1.0.9 del componente emisor de telemetría de núcleo.	23 de agosto de 2024
Versión 2.3.4 del administrador de Lambda publicada	Ya está disponible la versión 2.3.4 del componente administrador de Lambda.	23 de agosto de 2024
Versión 2.13.0 de la CLI de Greengrass publicada	Ya está disponible la versión 2.13.0 del componente de la CLI de Greengrass.	23 de agosto de 2024
Versión 2.5.1 del componente de autenticación del dispositivo de cliente publicada	Ya está disponible la versión 2.5.0 del componente de autenticación del dispositivo de cliente. Este lanzamiento suma compatibilidad con el punto de conexión de FIPS.	23 de agosto de 2024

Validación de receta	Se agregó una característica de validación de recetas que validará la receta de un componente al crear una versión del componente.	15 de agosto de 2024
Versión 2.2.0 del detector de IP publicada	Ya está disponible la versión 2.2.0 del component e detector de IP. Esta versión añade compatibilidad con. IPv6 Ahora puede usarlo IPv6 para mensajería local.	29 de julio de 2024
Versión 2.3.8 del administr ador de sombras publicada	Ya está disponible la versión 2.3.8 del administrador de sombras. Este lanzamien to corrige un problema que provocaba que el administr ador de sombras creara una situación de bloqueo durante la conexión del cliente MQTT.	5 de junio de 2024
Versión 2.12.6 de la CLI de Greengrass publicada	Ya está disponible la versión 2.12.6 del componente de la CLI de Greengrass.	24 de mayo de 2024
AWS IoT Greengrass Actualización de software Core v2.12.6	Esta versión proporciona la versión 2.12.6 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	24 de mayo de 2024

AWS IoT Device Tester Se ha publicado la versión 4.9.4 con la versión 2.5.4 de Q GGV2	Está disponible la versión 4.9.4 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.5.4 y es compatible con las versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 de Greengrass nucleus.	3 de mayo de 2024
Versión 1.0.19 de tunelización segura publicada	Ya está disponible la versión 1.0.19 de tunelización segura. Esta versión actualiza el Device Client de AWS IoT subyacente invocado por el componente de la versión 1.8.0 a la versión 1.9.0. La versión 1.0.19 de tunelización segura aumenta el límite de túneles simultáneos a 20 túneles a nivel de componente. Esta nueva versión también aumenta AWS IoT Greengrass el tiempo de espera del Core IPC de 3 a 10 segundos.	1 de mayo de 2024
Versión 1.0.5 del conector periférico para el componente Kinesis Video Streams publicada	Está disponible la versión 1.0.5 del conector periférico para el componente Kinesis Video Streams. Esta versión incluye correcciones de errores y mejoras generales.	29 de abril de 2024

Versión 2.12.5 de la CLI de Greengrass publicada	Ya está disponible la versión 2.12.5 del componente de la CLI de Greengrass.	25 de abril de 2024
Versión 2.5.0 del componente de autenticación del dispositivo de cliente publicada	Ya está disponible la versión 2.5.0 del componente de autenticación del dispositivo de cliente. Este lanzamiento suma compatibilidad con variables de política para los nombres de los objetos. Este lanzamiento también permite usar caracteres comodín para los recursos de políticas.	25 de abril de 2024
AWS IoT Greengrass Actualización del software Core v2.12.5	Esta versión incluye la versión 2.12.5 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	25 de abril de 2024
AWS IoT Device Tester Se ha publicado la versión 4.9.3 con la versión 2.5.3 de Q GGV2	Está disponible la versión 4.9.3 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.5.3 y es compatible con las versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 de Greengrass nucleus.	5 de abril de 2024
Versión 2.12.4 de la CLI de Greengrass publicada	Ya está disponible la versión 2.12.4 del componente de la CLI de Greengrass.	2 de abril de 2024

AWS IoT Greengrass Actualización de software Core v2.12.4	Esta versión proporciona la versión 2.12.4 del componente núcleo de Greengrass y actualiza los componentes proporcionados. AWS	2 de abril de 2024
Versión 2.3.7 del administrador de sombras publicada	Ya está disponible la versión 2.3.7 del administrador de sombras. Este lanzamiento corrige un problema por el que el administrador de sombras registra periódicamente un error `NullPointerException` durante una sincronización del administrador de sombras.	27 de marzo de 2024
Versión 2.3.6 del agente MQTT 3.1.1 de Moquette publicada	Ya está disponible la versión 2.3.6 del componente agente MQTT 3.1.1 de Moquette. Esta versión incluye correcciones de errores y mejoras generales.	27 de marzo de 2024
Versión 2.4.2 de la consola de depuración local publicada	Ya está disponible la versión 2.4.2 del componente de la consola de depuración local. Esta versión incluye correcciones de errores y mejoras generales.	27 de marzo de 2024
Versión 2.3.3 del administrador de Lambda publicada	Ya está disponible la versión 2.3.3 del componente administrador de Lambda. Esta versión incluye correcciones de errores y mejoras generales.	27 de marzo de 2024

Versión 2.1.9 del detector de IP publicada	Ya está disponible la versión 2.1.9 del componente detector de IP. Este lanzamien to ajusta el paso de IP adquirido para enviar solo los registros a nivel del registro de depuración.	27 de marzo de 2024
AWS IoT Lanzamiento del complemento de aprovisio namiento de flotas, versión 1.2.1	AWS IoT Está disponible la versión 1.2.1 del complemen to de aprovisionamiento de flotas. Este lanzamiento corrige un problema por el que el complemento de aprovisio namiento de flota no estaba en línea durante el arranque del núcleo de Greengrass. El complemento de aprovisio namiento de flotas ahora reintenta indefinidamente las llamadas de conexión MQTT.	27 de marzo de 2024
AWS IoT Greengrass Actualización de software Core v2.12.3	Esta versión proporciona la versión 2.12.3 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	27 de marzo de 2024
Versión 2.12.3 de la CLI de Greengrass publicada	Ya está disponible la versión 2.12.3 del componente de la CLI de Greengrass.	25 de marzo de 2024

AWS IoT Device Tester Se ha publicado la versión 4.9.2 con la versión 2.5.2 de Q GGV2	Está disponible la versión 4.9.2 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificac ión AWS IoT Greengrass V2 (GGV2Q) v2.5.2 y es compatible con las versiones 2.12.0, 2.11.0, 2.10.0 y 2.9.5 de Greengrass nucleus.	18 de marzo de 2024
AWS IoT Greengrass Actualización de software Core v2.12.2	Esta versión proporciona la versión 2.12.2 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	15 de febrero de 2024
Versión 2.3.6 del administr ador de sombras publicada	Ya está disponible la versión 2.3.6 del administr ador de sombras. Esta versión corrige un problema por el que las propiedades ocultas que se eliminan durante Nube de AWS las actualizaciones mientras el dispositivo está fuera de línea siguen existiend o en la sombra local después de recuperar la conectividad.	14 de febrero de 2024
Versión 2.0.13 del lanzador de Lambda publicada	Ya está disponible la versión 2.0.13 del component e lanzador de Lambda. Este lanzamiento incluye mejoras generales de rendimiento y correcciones de errores.	14 de febrero de 2024

Versión 1.0.3 del spooler de disco publicada	Ya está disponible la versión 1.0.3 del spooler de disco. Este lanzamien to mejora el rendimiento al reutilizar las conexiones de bases de datos.	14 de febrero de 2024
Versión 1.6.2 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.6.2 de la CLI del kit de desarrollo de Greengras s. Esta versión corrige un problema por el que el archivo gradlew.bat de Windows no funcionaba debido a la ruta relativa. Esta versión también contiene mejoras adicionales.	16 de enero de 2024
Nuevos eventos CloudTrail de datos	Ahora puede registrar eventos de AWS CloudTrail datos para obtener información sobre las operaciones de recursos, como la obtención de un componente o la configura ción de una implementación. Utilice estos eventos para obtener información sobre el funcionamiento de sus dispositivos de Greengrass.	20 de diciembre de 2023
Versión 2.1.12 del administr ador de flujos publicada	Ya está disponible la versión 2.1.12 del administr ador de flujos. Esta versión cambia el orden que Greengrass utiliza para seleccionar un conjunto de credenciales para las llamadas de AWS servicio.	8 de diciembre de 2023

Versión 2.3.1 del puente de MQTT publicada	Ya está disponible la versión 2.3.1 del puente de MQTT. Este lanzamiento corrige un problema poco frecuente que provocaba que el cliente de MQTT local entrara en un bucle de desconexión.	8 de diciembre de 2023
Versión 1.0.2 del spooler de disco publicada	Ya está disponible la versión 1.0.2 del spooler de disco. Este lanzamien to corrige un problema por el que el campo de formato de mensaje MQTT no se conserva en algunos casos.	8 de diciembre de 2023
Versión 2.4.5 del componente de autenticación del dispositi vo de cliente publicada	Ya está disponible la versión 2.4.5 del componente de autenticación del dispositi vo de cliente. Este lanzamien to suma compatibilidad con los caracteres comodín al final de los nombres de los objetos en una regla de selección y corrige un problema por el que, en algunos casos, los certificados no se actualiza ban con nueva información de conectividad.	8 de diciembre de 2023
AWS IoT Greengrass Actualización de software Core v2.12.1	Esta versión proporciona la versión 2.12.1 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	8 de diciembre de 2023

Versión 1.6.1 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.6.1 de la CLI del kit de desarrollo de Greengras s. Esta versión contiene correcciones de errores y mejoras.	6 de diciembre de 2023
Validación de receta	Se agregó una característica de validación de recetas que validará la receta de un componente al crear una versión del componente.	16 de noviembre de 2023
Componentes compatibles con Publisher	AWS IoT Greengrass ahora ofrece componentes compatibl es con Publisher. Proveedores externos desarrollan, ofrecen y mantienen estos component es.	16 de noviembre de 2023
Versión 1.2.0 de Greengrass Testing Framework publicada	Ya está disponible la versión 1.2.0 de Greengrass Testing Framework.	15 de noviembre de 2023

Versión 1.6.0 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.6.0 de la CLI del kit de desarrollo de Greengras s. Esta versión agrega una comprobación de validació n de receta con respecto al esquema de receta de Greengrass durante los comandos `component build` y `component publish`. Esta actualiza ción ayuda a los desarroll adores a identificar problemas procesables en sus recetas de componentes en una fase temprana del proceso de creación de los componentes. Esta versión también agrega un conjunto de pruebas de confianza a la plantilla que se puede implementar con el comando `test-e2e init`. Este conjunto de pruebas de confianza incluye ocho pruebas genéricas que se pueden utilizar y ampliar para adaptarse a las necesidad es básicas de las pruebas de componentes.	15 de noviembre de 2023
AWS IoT Device Tester La versión 4.9.1 es compatibl e con la versión 2.12.0 del núcleo de Greengrass	La versión 4.9.1 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.12.0 del núcleo de Greengrass.	7 de noviembre de 2023

AWS IoT Greengrass Actualización de software Core v2.12.0	Esta versión proporciona la versión 2.12.0 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	7 de noviembre de 2023
Operación de un dispositivo principal de Greengrass en la VPC	Ya está disponible la operación de un dispositivo principal de Greengrass en la VPC. Esta característica le permite realizar implement aciones en la VPC sin acceso público a Internet.	3 de noviembre de 2023
Versión 2.12.0 de la CLI de Greengrass publicada	Ya está disponible la versión 2.12.0 del componente de la CLI de Greengrass.	30 de octubre de 2023
Versión 2.1.10 del administr ador de flujos publicada	Ya está disponible la versión 2.1.10 del administr ador de flujos. Este lanzamien to corrige un problema por el que la configuración del proxy HTTPS no confiaba en la cadena de certificados de la CA de Greengrass.	26 de octubre de 2023
Versión 2.0.12 del lanzador de Lambda publicada	Ya está disponible la versión 2.0.12 del component e lanzador de Lambda. Este lanzamiento corrige un problema por el que el lanzador de Lambda podía generar un error si el proceso anterior no se detenía correctamente.	26 de octubre de 2023

Versión 1.5.0 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.5.0 de la CLI del kit de desarrollo de Greengras s. Esta versión actualiza los patrones reconocidos por la opción de compilaci ón de `excludes` cuando `build_system` está `zip`. Esta versión ahora reconocer á los patrones globales que coincidan con los nombres de las rutas en función de sus caracteres comodín. Esto permite especificar de forma personalizada los directorios de los que se debe excluir.	26 de octubre de 2023
Versión 2.3.4 del administr ador de sombras publicada	Ya está disponible la versión 2.3.4 del administrador de sombras. Este lanzamien to suma compatibilidad con documentos de estado de sombra nulos y vacíos.	18 de octubre de 2023
Versión 2.3.6 del administr ador de registros publicada	Ya está disponible la versión 2.3.6 del componente del administrador de registros.	18 de octubre de 2023
Versión 2.4.0 de la consola de depuración local publicada	Ya está disponible la versión 2.4.0 del component e de la consola de depuración local.	18 de octubre de 2023
Versión 2.3.1 del administr ador de Lambda publicada	Ya está disponible la versión 2.3.1 del componente administrador de Lambda.	18 de octubre de 2023

Versión 2.11.3 de la CLI de Greengrass publicada	Ya está disponible la versión 2.11.3 del componente de la CLI de Greengrass.	18 de octubre de 2023
AWS IoT Greengrass Actualización del software Core v2.11.3	Esta versión proporciona la versión 2.11.3 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	18 de octubre de 2023
Versión 1.0.17 de tunelización segura publicada	Ya está disponible la versión 1.0.17 de tunelización segura.	4 de octubre de 2023
Versión 1.4.0 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.4.0 de la CLI del kit de desarrollo de Greengras s. Esta versión agrega un nuevo comando `config` que inicia una petición interacti va para modificar los campos de un archivo de configura ción del GDK existente. Esta versión también modifica los comandos `gdk component build` y `gdk component publish` para comprobar que el tamaño de la receta cumple con los requisitos de Greengrass (<=16000 bytes) antes de continuar.	2 de octubre de 2023

Versión 2.3.5 del agente MQTT 3.1.1 de Moquette publicada	Ya está disponible la versión 2.3.5 del componente agente MQTT 3.1.1 de Moquette. Esta versión actualiza Moquette a la versión 0.17.	28 de septiembre de 2023
Versión 2.3.0 del puente de MQTT publicada	Ya está disponible la versión 2.3.0 del agente MQTT. Este lanzamiento suma compatibilidad con MQTT 5 para establecer conexiones entre AWS IoT Core y orígenes MQTT locales.	28 de septiembre de 2023
Versión 2.3.0 del administrador de Lambda publicada	Ya está disponible la versión 2.3.0 del componente administrador de Lambda.	15 de septiembre de 2023
Versión 2.0.11 del lanzador de Lambda publicada	Ya está disponible la versión 2.0.11 del componente lanzador de Lambda. Esta versión es compatible con la versión 2.3.0 del administrador de Lambda.	15 de septiembre de 2023
Versión 2.3.4 del agente MQTT 3.1.1 de Moquette publicada	Ya está disponible la versión 2.3.4 del componente agente MQTT 3.1.1 de Moquette.	1 de septiembre de 2023

Greengrass Testing Framework	El GTF es un conjunto de componentes básicos que respaldan la automatización. end-to-end Permite a los clientes internos de AWS IoT Greengrass Version 2 utilizar el mismo marco de pruebas que utiliza el equipo de servicio para calificar los cambios de software, aceptarlos automáticamente y garantizar la calidad.	11 de agosto de 2023
AWS IoT Greengrass Actualización de software Core v2.11.2	Esta versión proporciona la versión 2.11.2 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	9 de agosto de 2023
Versión 1.3.0 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.3.0 de la CLI del kit de desarrollo de Greengras s. Esta versión añade un nuevo `test-e2e` comando para permitir las end-to-en d pruebas de component es mediante Open Test Framework.	21 de julio de 2023
AWS IoT Greengrass Actualización de software Core v2.11.1	Esta versión proporciona la versión 2.11.1 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	21 de julio de 2023

Versión 1.0.0 del spooler de disco publicada	Ya está disponible la versión 1.0.0 del spooler de disco.	28 de junio de 2023
AWS IoT Greengrass Actualización de software Core v2.11.0	Esta versión proporciona la versión 2.11.0 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	28 de junio de 2023
AWS IoT Greengrass Actualización del software Core v2.10.3	Esta versión proporciona la versión 2.10.3 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	21 de junio de 2023
AWS IoT Greengrass Actualización de software Core v2.10.2	Esta versión proporciona la versión 2.10.2 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	5 de junio de 2023
AWS IoT Greengrass Actualización de software Core v2.10.1	Esta versión proporciona la versión 2.10.1 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados.	11 de mayo de 2023
AWS IoT Greengrass Actualización de software Core v2.10.0	Esta versión proporciona la versión 2.10.0 del component e núcleo de Greengrass y actualiza los componentes proporcionados. AWS	9 de mayo de 2023
SageMaker AI Edge Manager ha dejado de fabricarse	El componente Amazon SageMaker AI Edge Manager dejará de funcionar el 26 de abril de 2024.	28 de abril de 2023

AWS IoT Greengrass Actualización de software Core v2.9.6	Esta versión incluye la versión 2.9.6 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	20 de abril de 2023
Versión 2.3.2 del administrador de registros publicada	Ya está disponible la versión 2.3.2 del componente del administrador de registros.	19 de abril de 2023
Versión 2.1.4 del administrador de flujos publicada	Ya está disponible la versión 2.1.4 del administrador de flujos. Esta versión corrige un problema por el que las entradas del mismo activo inmobiliario con la misma marca de tiempo dentro de un mismo lote se devuelven `ConflictingOperationException` desde la SiteWise API, lo que provoca que Stream Manager vuelva a intentarlo continuamente. Este lanzamiento también actualiza el tiempo de espera de conexión predeterminado de 3 segundos a 1 minuto.	13 de abril de 2023
Versión 1.2.3 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.2.3 de la CLI del kit de desarrollo de Greengrass. Esta versión contiene correcciones de errores.	13 de abril de 2023

Versión 2.4.0 del componente de autenticación del dispositivo de cliente publicada	Ya está disponible la versión 2.4.0 del componente de autenticación del dispositivo de cliente. Este lanzamiento suma compatibilidad con la autenticación de dispositivos de cliente para emitir métricas operativas que se pueden mostrar en el panel de dispositivos de cliente de Greengrass.	10 de abril de 2023
Versión 1.2.2 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.2.2 de la CLI del kit de desarrollo de Greengrass. Esta versión contienen mejoras y correcciones de errores.	7 de abril de 2023
AWS IoT Greengrass Actualización de software Core v2.9.5	Esta versión incluye la versión 2.9.5 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	30 de marzo de 2023
Versión 2.1.3 del administrador de flujos publicada	Ya está disponible la versión 2.1.3 del administrador de flujos. Este lanzamiento corrige un problema de inicio en el sistema operativo Windows cuando se ejecuta como usuario SYSTEM.	7 de marzo de 2023

Versión 2.1.5 del adaptador de protocolo Modbus-RTU publicada	Ya está disponible la versión 2.1.5 del componente adaptador de protocolo Modbus-RTU. Este lanzamiento corrige un problema con la operación `ReadDiscreteInput` .	7 de marzo de 2023
Versión 2.3.2 del componente de autenticación del dispositivo de cliente publicada	Ya está disponible la versión 2.3.2 del componente del autenticación de dispositivos de cliente. Este lanzamiento suma compatibilidad con el almacenamiento en caché de la información del nombre de host, de modo que el componente genere correctamente los sujetos del certificado cuando se reinicie sin conexión a Internet.	7 de marzo de 2023
AWS IoT Device Tester La versión 4.7.0 es compatible con la versión 2.9.4 del núcleo de Greengrass	La versión 4.7.0 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.9.4 del núcleo de Greengrass.	2 de marzo de 2023
Versión 1.2.0 de la interfaz de la línea de comandos de Greengrass publicada	Ya está disponible la versión 1.2.0 de la interfaz de la línea de comandos de Greengrass.	28 de febrero de 2023
AWS IoT Greengrass Actualización de software Core v2.9.4	Esta versión incluye la versión 2.9.4 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	24 de febrero de 2023

Versión 2.3.1 del administrador de sombras publicada	Ya está disponible la versión 2.3.1 del administrador de sombras. Este lanzamiento corrige una condición que podía impedir la sincronización de las actualizaciones de la sombra en la nube. Este lanzamiento también corrige un problema por el que los cambios en la configuración de sincronización de sombras con nombre se aplicaban solo a una sombra con nombre.	21 de febrero de 2023
AWS IoT Device Tester La versión 4.7.0 es compatible con la versión 2.9.3 del núcleo de Greengrass	La versión 4.7.0 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.9.3 del núcleo de Greengrass.	9 de febrero de 2023
Prácticas recomendadas de IAM actualizadas	Se ha actualizado la guía para implementar las prácticas recomendadas de IAM. Para obtener más información, consulta prácticas recomendadas de seguridad en IAM.	3 de febrero de 2023
AWS IoT Greengrass Actualización de software Core v2.9.3	Esta versión incluye la versión 2.9.3 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	1 de febrero de 2023
Versión 2.3.1 del administrador de registros publicada	Ya está disponible la versión 2.3.1 del administrador de registros.	27 de enero de 2023

AWS IoT Device Tester La versión 4.7.0 es compatible con la versión 2.9.2 del núcleo de Greengrass	La versión 4.7.0 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.9.2 del núcleo de Greengrass.	3 de enero de 2023
Versión 2.3.0 del administrador de sombras publicada	Ya está disponible la versión 2.3.0 del administrador de sombras. Esta versión corrige un problema que podía impedir que las sombras se sincronizaran cuando un dispositivo guarda la clave privada del dispositivo de Greengrass en un módulo de seguridad de hardware.	29 de diciembre de 2022
AWS IoT Lanzamiento del complemento de aprovisionamiento de flotas v1.2.0	AWS IoT Está disponible el complemento de aprovisionamiento de flotas v1.2.0. Este lanzamiento suma compatibilidad con el aprovisionamiento de dispositivos mediante una solicitud de firma de certificado con una ruta de clave privada configurable.	22 de diciembre de 2022
AWS IoT Greengrass Actualización de software Core v2.9.2	Esta versión incluye la versión 2.9.2 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	22 de diciembre de 2022

AWS IoT Device Tester Se ha publicado la versión 4.7.0 con la versión 2.5.0 de Q GGV2	Está disponible la versión 4.7.0 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.5.0 y es compatible con las versiones 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 y 2.6.0 de Greengrass nucleus.	13 de diciembre de 2022
Versión 2.2.4 del administrador de sombras publicada	Soluciona un problema por el que la validación del tamaño de la sombra no era coherente con el de la nube al actualizar el documento de sombra local. Esto también soluciona un problema por el que el administrador de sombras deja de escuchar las actualizaciones de la configuración si una implementación realiza un RESET en los nodos de configuración.	8 de diciembre de 2022
Versión 2.3.0 del administrador de registros publicada	Ya está disponible la versión 2.3.0 del componente del administrador de registros.	18 de noviembre de 2022
AWS IoT Device Tester La versión 4.5.11 es compatible con la versión 2.9.1 del núcleo de Greengrass	La versión 4.5.11 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.9.1 del núcleo de Greengrass.	18 de noviembre de 2022

AWS IoT Greengrass Actualización de software Core v2.9.1	Esta versión incluye la versión 2.9.1 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	18 de noviembre de 2022
AWS IoT Device Tester La versión 4.5.11 es compatible con la versión 2.9.0 del núcleo de Greengrass	La versión 4.5.11 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.9.0 del núcleo de Greengrass.	17 de noviembre de 2022
Versión 2.1.2 del administrador de flujos publicada	Ya está disponible la versión 2.1.2 del administrador de flujos. Este lanzamiento corrige un problema en el que el sistema operativo Windows utilizaba un idioma distinto del inglés.	15 de noviembre de 2022
AWS IoT Greengrass Actualización de software Core v2.9.0	Esta versión proporciona la versión 2.9.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	15 de noviembre de 2022
AWS IoT Device Tester La versión 4.5.11 es compatible con la versión 2.8.1 del núcleo de Greengrass	La versión 4.5.11 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.8.1 del núcleo de Greengrass.	19 de octubre de 2022

AWS IoT Device Tester Publicada la versión 4.5.11 con Q, versión 2.4.1 GGV2	Está disponible la versión 4.5.11 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificac ión AWS IoT Greengrass V2 (GGV2Q) v2.4.1 y es compatible con las versiones 2.8.0, 2.7.0 y 2.6.0 del núcleo de Greengrass.	13 de octubre de 2022
AWS IoT Greengrass Actualización de software Core v2.8.1	Esta versión incluye la versión 2.8.1 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	13 de octubre de 2022
AWS IoT Greengrass Actualización de software Core v2.8.0	Esta versión proporciona la versión 2.8.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él.	7 de octubre de 2022
Se agregó soporte CloudForm ation para despliegues	CloudFormation ahora admite AWS IoT Greengrass las implementaciones como recurso.	6 de octubre de 2022

SageMaker Lanzamiento de AI Edge Manager v1.3.0	Está disponible el componente Amazon SageMaker AI Edge Manager v1.3.0. Esta versión añade compatibilidad con este componente para establecer el tamaño del disco para la caché del modelo TensorRT y mejora la simultaneidad de predicciones para aprovechar mejor los motores aceleradores de dispositivos, como. GPUs	1 de septiembre de 2022
Uso del cliente de comunicación entre procesos (IPC) V2	Se agregó información sobre la versión 2 del cliente de IPC, lo que reduce la cantidad de código que hay que escribir para utilizar las operaciones de IPC y ayuda a evitar los errores habituales que pueden producirse con la versión 1 del cliente de IPC.	12 de agosto de 2022
AWS IoT Device Tester Se lanzó la versión 4.5.8 con la versión 2.4.0 de Q GGV2	Está disponible la versión 4.5.8 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.4.0 y es compatible con las versiones 2.7.0, 2.6.0 y 2.5.6 del núcleo de Greengrass.	12 de agosto de 2022

SageMaker Lanzamiento de AI Edge Manager v1.2.0	Está disponible el componente Amazon SageMaker AI Edge Manager v1.2.0. Esta versión añade compatibilidad con este componente para recuperar automáticamente los modelos compilados por SageMaker AI NEO que usted carga en Amazon S3, de forma que pueda implementar nuevos modelos sin necesidad de crear una AWS IoT Greengrass implementación.	3 de agosto de 2022
AWS IoT Device Tester La versión 4.5.3 es compatible con la versión 2.7.0 del núcleo de Greengrass	La versión 4.5.3 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.7.0 del núcleo de Greengrass.	1 de agosto de 2022
Versión 2.1.0 del administrador de flujos publicada	Ya está disponible la versión 2.1.0 del administrador de flujos. Esta versión incluye soporte para enviar métricas de telemetría a Amazon. EventBridge	28 de julio de 2022
AWS IoT Greengrass Actualización de software Core v2.7.0	Esta versión incluye la versión 2.7.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye soporte para enviar métricas de telemetría a Amazon. EventBridge	28 de julio de 2022

Publicada la versión 2.2.0 de IoT SiteWise Publisher	Está disponible el componente IoT SiteWise Publisher v2.2.0. Esta versión actualiza el componente para comprimir los datos antes de enviarlos al AWS IoT SiteWise servicio, lo que reduce el uso de ancho de banda hasta en un 75 por ciento.	19 de julio de 2022
Tutorial: Desarrollo de un componente que interactúe con las sombras de dispositivo de cliente	Se agregó un nuevo módulo al Tutorial: Interacción con dispositivos IoT locales a través de MQTT que puede seguir para aprender a desarrollar un componente que interactúe con las sombras de dispositivo de cliente.	18 de julio de 2022
Elección de un agente MQTT local	Se agregó información sobre cómo elegir un agente MQTT local donde los dispositivos de cliente se conecten a un dispositivo principal.	18 de julio de 2022
AWS IoT Device Tester La versión 4.5.3 es compatible con la versión 2.6.0 del núcleo de Greengrass	La versión 4.5.3 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.6.0 del núcleo de Greengrass.	29 de junio de 2022

AWS IoT Greengrass Actualización de software Core v2.6.0	Esta versión proporciona la versión 2.6.0 del component e núcleo de Greengrass y AWS actualiza los component es proporcionados. Incluye compatibilidad para sombras de dispositivo de cliente y un agente MQTT 5 local para dispositivos de cliente. También admite caractere s comodín en los publish/ subscribe temas locales, variables de receta en las configuraciones de los componentes y caracteres comodín en las políticas de autorización del IPC. Estas características le permiten desarrollar y configurar más fácilmente los component es que se implementan en las flotas de dispositivos principales. Este lanzamien to también incluye compatibi lidad para que los component es utilicen operaciones de IPC que administran las implementaciones locales y los componentes de un dispositivo principal.	27 de junio de 2022

Actualizaciones de los componentes del dispositivo de cliente	Están disponibles la versión 2.1.0 de la autenticación de dispositivos de cliente, la versión 2.1.0 del agente MQTT (Moquette), la versión 2.1.1 del puente de MQTT y la versión 2.1.2 del detector de IP. Este lanzamiento mejora la rotación de certificados, mejora el rendimiento del agente de MQTT y corrige los problemas relacionados con la forma en que estos componentes gestionan las actualizaciones de restablecimiento de la configuración.	14 de junio de 2022
AWS IoT Device Tester La versión 4.5.3 es compatible con la versión 2.5.6 del núcleo de Greengrass	La versión 4.5.3 de IDT para AWS IoT Greengrass V2 ahora es compatible con la versión 2.5.6 del núcleo de Greengrass.	1 de junio de 2022
AWS IoT Greengrass Actualización de software Core v2.5.6	Esta versión incluye la versión 2.5.6 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye compatibilidad para módulos de seguridad de hardware con claves ECC. También incluye otras correcciones de errores y mejoras.	31 de mayo de 2022

AWS IoT Lanzamiento del complemento de aprovisionamiento de flotas, versión 1.1.0	AWS IoT Está disponible la versión 1.1.0 del complemento de aprovisionamiento de flotas. Este lanzamiento suma compatibilidad con otros formatos de rutas de archivos al configurar el complemento en dispositivos Windows.	12 de mayo de 2022
Nuevos tiempos de ejecución de Lambda publicados	Se agregó compatibilidad con los nuevos tiempos de ejecución de Lambda: Python 3.9, Java 11 y NodejS 14.	10 de mayo de 2022
Desarrollo de un componente de Greengrass que aplace las actualizaciones de los componentes	Se agregó un tutorial que puede seguir para aprender a desarrollar un componente de Greengrass que aplace las actualizaciones de componentes de las implementaciones. Es posible que desee retrasar una actualización cuando el nivel de batería de un dispositivo sea bajo o cuando ejecute un proceso que no pueda interrumpirse, por ejemplo.	4 de mayo de 2022

CloudWatch Publicadas las versiones 3.1.0 y 3.1.0 de metrics AWS IoT Device Defender	CloudWatch están disponibl es el componente de métricas v3.1.0 y el componente v3.1.0. AWS IoT Device Defender Estos lanzamientos agregan compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Conectarse en el puerto 443 o mediante un proxy de red y Habilitar el dispositivo principal para que confíe en un proxy HTTPS.	27 de abril de 2022
Migre desde AWS IoT Greengrass Version 1	Se agregó una guía que puede seguir para migrar de AWS IoT Greengrass V1 a AWS IoT Greengrass V2.	26 de abril de 2022

AWS IoT Device Tester v4.5.3 con GGV2 Q v2.3.1 actualizada e IDT v4.5.1 con Q v2.3.0 agregadas a las versiones compatibles GGV2	La versión 4.5.3 de IDT para AWS IoT Greengrass V2 con la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.3.1 se ha actualizado para incluir compatibilidad con las versiones 2.5.5, 2.5.4 y 2.5.3 de Greengrass nucleus. Esta actualización también incluye IDT 4.5.1 con la suite de calificación AWS IoT Greengrass V2 (Q) v2.3.0 como versión compatible. GGV2 IDT 4.5.1 con el paquete de calificación AWS IoT Greengrass V2 (GGV2Q) v2.3.0 es compatible con Greengrass nucleus versión 2.5.3.	25 de abril de 2022
Versión 2.1.0 del adaptador de protocolo Modbus-RTU publicada	Ya está disponible la versión 2.1.0 del componente adaptador de protocolo Modbus-RTU. Este lanzamiento agrega nuevos parámetros que puede especificar para configurar la comunicación en serie con los dispositivos Modbus RTU.	20 de abril de 2022

CloudWatch Lanzamiento de metrics v2.1.0, Firehose v2.1.0 y Amazon SNS v2.1.0	CloudWatch Están disponibles el componente de métricas v2.1.0, el componente Firehose v2.1.0 y el componente Amazon SNS v2.1.0. Estos lanzamientos agregan compatibilidad con las configuraciones de proxy de red HTTPS. Para obtener más información, consulte Conectarse en el puerto 443 o mediante un proxy de red y Habilitar el dispositivo principal para que confíe en un proxy HTTPS.	19 de abril de 2022
AWS IoT Device Tester SeGGV2 ha publicado la versión 4.5.3 con la versión 2.3.1 de Q	Está disponible la versión 4.5.3 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.3.1 y es compatible con la versión 2.5.5 del núcleo de Greengrass.	15 de abril de 2022

AWS IoT Greengrass Actualización de software Core v2.5.5	Esta versión incluye la versión 2.5.5 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Suma compatibilidad con dispositivos Windows que utilizan un idioma de visualización distinto del inglés. También corrige un problema por el que el dispositivo principal no informaba de su estado al servicio en la AWS IoT Greengrass nube después del aprovisionamiento en determinadas situaciones.	6 de abril de 2022
AWS IoT Greengrass Actualización del software Core v2.5.4	Esta versión incluye la versión 2.5.4 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye correcciones de errores y mejoras.	23 de marzo de 2022
AWS IoT Device Tester Descargue mediante programación	Se agregó información sobre cómo descargar IDT mediante programación. AWS IoT Greengrass V2	15 de marzo de 2022

Versión 1.1.0 de la CLI del kit de desarrollo de Greengrass	Ya está disponible la versión 1.1.0 de la CLI del kit de desarrollo de Greengras s. Esta versión agrega nuevos argumentos a los comandos `component init` y `component publish`. Esta versión también actualiza el comando `component publish` para compilar el componente si no está compilado.	24 de febrero de 2022
Versión 2.1.0 del administr ador de sombras publicada	Ya está disponible la versión 2.1.0 del component e administrador de sombras. Esta versión añade la opción de configurar el intervalo con el que el componente sincroniza las sombras. AWS IoT Core Por ejemplo, puede especificar un intervalo más largo para reducir el uso y los cargos del ancho de banda.	3 de febrero de 2022
Dockerfile e imágenes de Docker para la versión 2.5.3 del software Core AWS IoT Greengrass	Ya están disponibles el Dockerfile y la imagen de Docker para el software Core v2.5.3. AWS IoT Greengrass	12 de enero de 2022

AWS IoT Device Tester Se ha publicado la versión 4.5.1 con la versión 2.3.0 de Q GGV2	Está disponible la versión 4.5.1 de IDT para V2. AWS IoT Greengrass Esta versión incluye el paquete de calificación AWS IoT Greengrass V2 (GGV2Q) v2.3.0 y permite validar y calificar los dispositivos basados en Linux que utilizan un módulo de seguridad de hardware (HSM) para almacenar la clave privada y el certificado que utiliza el software Core. AWS IoT Greengrass	11 de enero de 2022
AWS IoT Greengrass Actualización de software Core v2.5.3	Esta versión incluye la versión 2.5.3 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye soporte para configurar el software AWS IoT Greengrass Core para que utilice una clave privada y un certificado que se almacenarán de forma segura en un módulo de seguridad de hardware (HSM).	6 de enero de 2022
Imágenes de Dockerfile y Docker para el software Core v2.5.2 AWS IoT Greengrass	Ya están disponibles el Dockerfile y la imagen de Docker para el software Core v2.5.2. AWS IoT Greengrass	20 de diciembre de 2021

AWS IoT Device Tester Se ha publicado la versión 4.4.1 con la versión 2.2.1 de Q GGV2	Está disponible la versión 4.4.1 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.2.1 y es compatible con la versión 2.5.2 de Greengrass nucleus para la calificación de dispositivos.	12 de diciembre de 2021
Realización de inferencias de machine learning con Amazon Lookout for Vision	Se agregó información sobre cómo realizar inferencias de machine learning con Lookout for Vision en los dispositivos principales de Greengrass. Lookout for Vision utiliza la visión de computadora para detectar defectos en productos industriales.	8 de diciembre de 2021
AWS IoT Device Tester Se ha publicado la versión 4.4.1 con la versión 2.2.0 de Q GGV2	Está disponible la versión 4.4.1 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificación AWS IoT Greengrass V2 (GGV2Q) v2.2.0 y es compatible con la versión 2.5.2 de Greengrass nucleus para la calificación de dispositivos.	6 de diciembre de 2021

AWS IoT Greengrass Actualización de software Core v2.5.2	Esta versión incluye la versión 2.5.2 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Corrige un problema con el servicio de Windows que se produce después de que se actualice el núcleo de Greengrass. También incluye soporte para el AWS IoT Device Defender componente en dispositivos Windows.	3 de diciembre de 2021
Nuevo conector periférico para el componente Kinesis Video Streams	Está disponible la versión 1.0.0 del conector periférico para el component e Kinesis Video Streams. Este componente con AWS lee las transmisiones de video de las cámaras locales y las publica en Kinesis Video Streams. Este componente se integra con AWS IoT TwinMaker, lo que le permite ver y administr ar las transmisiones de vídeo y otros datos en los paneles de Grafana.	30 de noviembre de 2021

Gestione los dispositivos principales de Greengrass con AWS Systems Manager	Se agregó información sobre cómo administrar los dispositivos principales de Greengrass con. AWS Systems Manager Systems Manager es un servicio de AWS que le permite ver los datos operativos, automatizar las tareas operativas y mantener la seguridad y el cumplimiento.	29 de noviembre de 2021
CLI del kit de desarrollo de Greengrass	Se agregó información sobre la interfaz de línea de comandos del kit de AWS IoT Greengrass desarrollo (GDK CLI), que es una herramienta que puede descargar en su computadora de desarrollo local para ayudarlo a desarrollar componentes personalizados de Greengrass. Puede usar la CLI del GDK para crear, compilar y publicar componentes personalizados.	29 de noviembre de 2021
Componentes de Greengrass proporcionados por la comunidad	Se agregó información sobre el catálogo de software de Greengrass, que es un índice de los componentes de Greengrass desarrollados por la comunidad de Greengrass. Desde este catálogo, puede descargar, modificar e implementar componentes para crear sus aplicaciones de Greengrass.	29 de noviembre de 2021

AWS IoT Greengrass Actualización del software Core v2.5.1	Esta versión incluye la versión 2.5.1 del component e núcleo de Greengrass y AWS actualiza los component es proporcionados por él. Incluye compatibilidad para Java de 32 bits en dispositi vos Windows. También corrige problemas relacionados con el nuevo comportamiento de eliminación de grupos de objetos y la carga de las variables de entorno del sistema en los dispositivos Windows.	23 de noviembre de 2021
AWS IoT Device Tester Publicada la versión 4.4.0 con la versión 2.1.0 de Q GGV2	Está disponible la versión 4.4.0 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificac ión AWS IoT Greengrass V2 (GGV2Q) v2.1.0 y admite la calificación de dispositi vos Greengrass basados en Windows que ejecutan Greengrass nucleus versión 2.5.0.	19 de noviembre de 2021

AWS IoT Greengrass Actualización de software Core v2.5.0	Esta versión proporciona la versión 2.5.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye soporte para ejecutar el software AWS IoT Greengrass Core en dispositivos Windows. También cambia el comportamiento de eliminación de grupos de objetos y suma compatibilidad con los proxies HTTPS.	12 de noviembre de 2021
SageMaker Lanzamiento de AI Edge Manager v1.1.0	Ya está disponible el componente Amazon SageMaker AI Edge Manager v1.1.0. Este lanzamiento suma compatibilidad con los dispositivos principales de Greengrass que ejecutan Amazon Linux 2 y agrega un nuevo parámetro de configuración para especificar la ubicación de la carpeta de datos de captura en el dispositivo.	3 de noviembre de 2021
Actualización de la prevención del suplente confuso entre servicios	AWS IoT Greengrass V2 admite el uso de las claves de contexto `aws:SourceArn` y las condiciones `aws:SourceAccount` globales en las políticas de recursos de IAM para evitar el confuso problema de los diputados.	1 de noviembre de 2021

Actualizaciones de los componentes del dispositivo de cliente	Ya está disponible la versión 2.0.3 de la autenticación de dispositivos de cliente, la versión 2.1.0 del detector de IP, la versión 2.1.0 del puente de MQTT y la versión 2.0.2 del agente MQTT (Moquette). Este lanzamiento suma compatibilidad total con los puertos de agente MQTT no predeterminados e incluye otras correcciones de errores y mejoras.	28 de octubre de 2021
Versión 2.0.4 del administrador de sombras publicada	Ya está disponible la versión 2.0.4 del componente administrador de sombras. Este lanzamiento corrige un problema que provocaba que el administrador de sombras eliminara las versiones recién creadas de cualquier sombra que se hubiera eliminado anteriormente. A partir de esta versión, la operación de IPC `DeleteThingShadow` incrementa la versión de sombra.	20 de octubre de 2021

Versión 2.2.0 del administrador de registros publicada	Ya está disponible la versión 2.2.0 del componente del administrador de registros . El administrador de registros ahora admite el uso de un mapa de configuración para proporcionar las configuraciones de registro de los componentes.	20 de octubre de 2021
Versión 2.1.4 del administrador de Lambda publicada	Ya está disponible la versión 2.1.4 del componente administrador de Lambda. Este lanzamiento corrige un problema que provocaba que las funciones de Lambda que utilizan tiempos de ejecución de NodeJS procesaran solo un mensaje.	20 de octubre de 2021
Utilice la comunicación entre procesos, AWS las credenciales y el administrador de flujos en los componentes del contenedor de Docker	Se agregó información sobre cómo usar la comunicación entre procesos (IPC), las credenciales de AWS y el administrador de flujos en sus componentes de contenedor de Docker personalizados.	19 de octubre de 2021

Nuevo componente emisor de telemetría de núcleo	Ya está disponible la versión 1.0.0 del component e emisor de telemetría del núcleo. Este componente AWS proporcionado recopila datos de telemetría del estado del sistema y los publica continuamente en un tema local y en un tema de MQTT. AWS IoT Core	30 de septiembre de 2021
Cómo permitir el tráfico del dispositivo a través de un proxy o firewall	Se agregó información sobre los puntos de conexión y puertos que utilizan los dispositivos principales de Greengrass, para que pueda restringir el tráfico como medida de seguridad.	16 de septiembre de 2021
AWS IoT Device Tester Versión 4.2.0 y versión 2.0.1 de Q GGV2	La versión 4.2.0 de IDT para la versión AWS IoT Greengras s 2 se ha actualizado con la versión 2.0.1 de la suite de calificación AWS IoT Greengrass V2 (Q). GGV2 Este lanzamiento es compatibl e con la versión 2.4.0 del núcleo de Greengrass para la calificación de dispositivos.	31 de agosto de 2021

Componentes del instalador de machine learning actualizados	Están disponibles el componente instalador DLR v1.6.5 y TensorFlow el componente instalador Lite v2.5.4. Estas versiones de componentes incluyen el nuevo parámetro de configuración `UseInstaller` que le permite deshabilitar el script de instalación predeterminado.	30 de agosto de 2021
Soporte integrado de Linux para AWS IoT Greengrass	La BitBake receta AWS IoT Greengrass V2 está disponible en el `meta-aws` proyecto en GitHub. Puede utilizar esta receta para crear un sistema operativo personalizado basado en Linux mediante el Proyecto Yocto.	20 de agosto de 2021
Integridad del código	Se agregó información sobre cómo se AWS IoT Greengrass V2 verifica la integridad del software que los dispositivos principales de Greengrass descargan desde. Nube de AWS	19 de agosto de 2021
Puntos de conexión de VPC (AWS PrivateLink)	AWS IoT Greengrass ahora admite puntos finales de VPC de interfaz (AWS PrivateLink) para el AWS IoT Greengrass plano de control. Puede establecer una conexión privada entre la VPC y el plano de AWS IoT Greengrass control.	16 de agosto de 2021

Versión 2.0.12 del administrador de flujos publicada	Ya está disponible la versión 2.0.12 del administrador de flujos. Este lanzamiento corrige un problema que impedía actualizar la versión 2.0.7 del componente administrador de flujos a una versión entre la 2.0.8 y la 2.0.11.	10 de agosto de 2021
Dockerfile e imágenes de Docker para el software Core v2.4.0 AWS IoT Greengrass	Ya están disponibles el Dockerfile y la imagen de Docker para el software Core v2.4.0. AWS IoT Greengrass	9 de agosto de 2021
AWS IoT Greengrass Actualización del software Core v2.4.0	Esta versión proporciona la versión 2.4.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye compatibilidad con los límites de recursos del sistema de componentes, las operaciones IPC para pausar y reanudar los componentes y el aprovisionamiento de complementos.	3 de agosto de 2021
Componentes nuevos AWS IoT SiteWise	Se agregaron los siguientes componentes AWS proporcionados para AWS IoT SiteWise: el recopilador SiteWise OPC-UA de IoT, el SiteWiseeditor de IoT y el procesador de IoT. SiteWise	29 de julio de 2021

AWS IoT Device Tester Lanzamiento de la versión 4.2.0 con Q de la versión 2.0.0 GGV2	Está disponible la versión 4.2.0 de IDT para V2. AWS IoT Greengrass Esta versión incluye el paquete de calificación AWS IoT Greengrass V2 (GGV2Q) v2.0.0 e incluye soporte para pruebas de calificación opcionales para los componentes de Docker, el aprendizaje automático y el administrador de transmisiones.	14 de julio de 2021
AWS IoT Greengrass La biblioteca IPC básica está disponible en C++ v2 SDK para dispositivos con AWS IoT	La versión 1.13.0 de la versión SDK para dispositivos con AWS IoT para C++ es compatible con AWS IoT Greengrass Core IPC, por lo que puede desarrollar componentes en C++ que interactúen con el software Core. AWS IoT Greengrass	14 de julio de 2021
SageMaker Lanzamiento del componente AI Edge Manager v1.0.2	El componente Amazon SageMaker AI Edge Manager v1.0.2 está disponible. Este lanzamiento actualiza el script de instalación en el ciclo de vida del componente. Sus dispositivos principales ahora deben tener Python 3.6 o posterior, incluida `pip` para su versión de Python, instalado en el dispositivo antes de implementar este componente.	12 de julio de 2021

Actualización de soporte AWS IoT Device Tester para AWS IoT Greengrass V2	La versión 4.1.0 de IDT para AWS IoT Greengrass V2 ahora admite el uso de la versión 2.3.0 del núcleo de Greengrass para la calificación de los dispositivos.	8 de julio de 2021
Dockerfile e imágenes de Docker para el software Core v2.3.0 AWS IoT Greengrass	Ya están disponibles el Dockerfile y la imagen de Docker para el software Core v2.3.0. AWS IoT Greengrass	7 de julio de 2021
AWS políticas gestionadas	Se agregó información sobre las políticas AWS administr adas para AWS IoT Greengras s.	2 de julio de 2021
Nuevas opciones de JVM recomendadas	Se agregó información sobre las opciones de JVM recomendadas para controlar la asignación de memoria para el software AWS IoT Greengrass Core.	30 de junio de 2021
AWS IoT Greengrass Actualización del software Core v2.3.0	Esta versión proporciona la versión 2.3.0 del component e núcleo de Greengrass y AWS actualiza los component es proporcionados por él. Incluye compatibilidad para documentos de configura ción de componentes de gran tamaño en las implement aciones.	29 de junio de 2021

Dockerfile e imágenes de Docker para la versión 2.2.0 del software Core AWS IoT Greengrass	Ya están disponibles el Dockerfile y la imagen de Docker para el software Core v2.2.0. AWS IoT Greengrass	28 de junio de 2021
AWS IoT Device Tester Se ha publicado la versión 4.1.0 con la versión 1.1.1 de Q GGV2	Está disponible la versión 4.1.0 de IDT para V2. AWS IoT Greengrass Esta versión incluye la suite de calificac ión AWS IoT Greengrass V2 (GGV2Q) v1.1.1 y admite el uso de Greengrass nucleus v2.2.0, v2.1.0 y v2.0.5 para la calificación de dispositivos.	18 de junio de 2021
AWS IoT Greengrass Actualización de software Core v2.2.0	Esta versión proporciona la versión 2.2.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye componentes que puede implementar para sumar compatibilidad a los dispositi vos de cliente y agregar el servicio de sombra local.	18 de junio de 2021
Versión 2.0.6 del lanzador de Lambda publicada	Ya está disponible la versión 2.0.6 del component e lanzador de Lambda. Esta versión contiene mejoras de rendimiento y correcciones de errores.	13 de junio de 2021

Lanzamiento del nuevo SageMaker componente AI Edge Manager	La versión 1.0.0 del componente Amazon SageMaker AI Edge Manager está disponible para AWS IoT Greengrass. Este componente instala el binario del agente SageMaker AI Edge Manager en los dispositivos principales de Greengrass.	10 de junio de 2021
Tipos de componentes	Se agregó información sobre los tipos de componentes en AWS IoT Greengrass. El tipo de componente especifica cómo el software AWS IoT Greengrass Core ejecuta un componente.	3 de junio de 2021
AWS IoT Device Tester Publicada la versión 4.0.2 con la versión 1.1.0 de Q GGV2	Está disponible la versión 4.0.2 de IDT para V2. AWS IoT Greengrass Esta versión incluye el paquete de calificación AWS IoT Greengrass V2 (GGV2Q) v1.1.0 y admite el uso de Greengrass nucleus v2.1.0 con Greengrass CLI v2.1.0 para la calificación de dispositivos. Esto también incluye los nuevos grupos de pruebas necesarios para MQTT y Lambda, así como otras correcciones de errores y mejoras menores.	5 de mayo de 2021

Imágenes de Dockerfile y Docker para el software Core v2.1.0 AWS IoT Greengrass	Ya están disponibles el Dockerfile y la imagen de Docker para el software Core v2.1.0. AWS IoT Greengras s La imagen de Docker le permite ejecutar el software AWS IoT Greengrass principal en un contenedor de Docker que usa Amazon Linux 2 como sistema operativo base.	27 de abril de 2021
AWS IoT Greengrass Actualización del software Core v2.1.0	Esta versión proporciona la versión 2.1.0 del componente núcleo de Greengrass y AWS actualiza los componentes proporcionados por él. Incluye un nuevo componente que puede usar para descargar imágenes de Docker desde repositorios privados de Amazon ECR y nuevos componentes de muestra para realizar inferencias de aprendizaje automático con Lite. TensorFlow	26 de abril de 2021
Ejemplo de componente que usa Secrets Manager	Se agregó un componente de ejemplo que imprime el valor de un AWS Secrets Manager secreto que se implementa en un dispositivo principal.	8 de abril de 2021

AWS IoT Política mínima para los dispositivos principales de Greengrass	Se agregó información sobre el conjunto mínimo de permisos necesarios para admitir la funcionalidad básica de Greengrass en un dispositivo principal.	2 de abril de 2021
Suscripción a los flujos de eventos de IPC	Se agregó información sobre cómo usar las operaciones de comunicación entre procesos (IPC) para suscribirse a flujos de eventos en un dispositivo principal de Greengrass.	1 de abril de 2021
Actualización de soporte AWS IoT Device Tester para AWS IoT Greengrass	La versión 4.0.1 de IDT para AWS IoT Greengrass V2 ahora admite el uso de la versión 2.0.5 del núcleo de Greengrass con la versión 2.0.5 de la CLI de Greengrass para la calificación de los dispositivos.	17 de marzo de 2021
Creación de componentes personalizados que usen el administrador de flujos	Se agregó información sobre cómo configurar las recetas de componentes y los artefactos para desarrollar aplicaciones que administren los flujos de datos.	9 de marzo de 2021

AWS IoT Greengrass Actualización de software Core v2.0.5	Esta versión proporciona la versión 2.0.5 del component e núcleo de Greengrass y AWS actualiza los component es proporcionados por él. Soluciona un problema con la compatibilidad con el proxy de red y un problema con el punto final del plano de datos de Greengrass en las regiones de AWS China.	9 de marzo de 2021
Referencia de variables de entorno de componentes	Se agregó información sobre las variables de entorno que el software AWS IoT Greengras s Core establece para los componentes. Puede usar estas variables de entorno para obtener el nombre de la cosa y la Región de AWS versión del núcleo de Greengrass.	23 de febrero de 2021

Instalación manual	Se agregó información sobre cómo crear AWS los recursos necesarios manualmente o instalarlos detrás de un firewall o un proxy de red. Al realizar una instalación manual, no es necesario dar permiso al instalador para crear recursos en la suya Cuenta de AWS, ya que crea los recursos necesarios AWS IoT y los de IAM. También puede configura r el dispositivo para que se conecte al puerto 443 o a través de un proxy de red.	17 de febrero de 2021
AWS IoT Greengrass Actualización de la biblioteca Core IPC SDK para dispositi vos con AWS IoT para Python v2	La versión 1.5.4 de la versión SDK para dispositivos con AWS IoT para Python simplific a los pasos necesarios para conectarse al servicio AWS IoT Greengrass Core IPC.	11 de febrero de 2021
Actualización de soporte AWS IoT Device Tester para AWS IoT Greengrass	La versión 4.0.1 de IDT para AWS IoT Greengrass V2 ahora admite el uso de la versión 2.0.4 del núcleo de Greengrass con la versión 2.0.4 de la CLI de Greengras s para la calificación de los dispositivos.	5 de febrero de 2021

Nuevo tutorial para importar funciones de Lambda	Se agregó un nuevo tutorial basado en la consola para importar una función de Lambda como un component e que se ejecuta en el dispositi vo principal de Greengrass.	5 de febrero de 2021
AWS IoT Greengrass Actualización de software Core v2.0.4	Este lanzamiento proporciona la versión 2.0.4 del component e de núcleo de Greengrass. Incluye el nuevo parámetro `greengrassDataPlan ePort` para configurar la comunicación HTTPS a través del puerto 443 y corrige errores. La política de IAM mínima ahora requiere que `iam:GetPolicy` y `sts:GetCallerIdent ity` cuando se ejecute el instalador del software AWS IoT Greengrass Core. `--provision true`	4 de febrero de 2021

Nuevo componente de tunelización segura publicado	La versión 1.0.0 del componente de tunelización segura está disponible para. AWS IoT Greengrass Este componente AWS proporcionado utiliza una tunelización AWS IoT segura para establecer una comunicación bidireccional segura con un dispositivo central de Greengrass que se encuentra detrás de firewalls restringidos.	21 de enero de 2021
AWS IoT Device Tester para la versión 4.0.1 publicada AWS IoT Greengrass	Ya está disponible la versión 4.0.1 de IDT para AWS IoT Greengrass la versión V2. Esta versión le permite usar IDT para desarrollar y ejecutar sus conjuntos de pruebas personalizadas para la validación de dispositivos. Esto también incluye aplicaciones IDT firmadas con código para macOS y Windows.	22 de diciembre de 2020

Versión inicial de AWS IoT Greengrass Version 2	AWS IoT Greengrass V2 es una nueva versión principal de AWS IoT Greengrass. Esta versión agrega varias características, como componentes de software modulares e implementaciones continuas. Estas características le facilitan el desarrollo y la administración de aplicaciones de periferia.	15 de diciembre de 2020

AWS Glosario

Para obtener la AWS terminología más reciente, consulte el AWS glosario de la Glosario de AWS Referencia.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.