



開發人員指南，第 2 版

AWS IoT Greengrass



AWS IoT Greengrass: 開發人員指南，第 2 版

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 AWS IoT Greengrass ?	1
新功能	1
適用於第一次使用的使用者	1
對於現有使用者	2
AWS IoT Greengrass 運作方式	2
重要概念	3
的功能 AWS IoT Greengrass	5
Greengrass 功能相容性	6
選擇您的 nucleus AWS IoT Greengrass 執行時間	14
Greengrass 核	15
Greengrass nucleus lite	15
第 2 版中的新功能	18
AWS IoT Greengrass Core v2.16.1 軟體更新	20
公有元件更新	20
AWS IoT Greengrass Core v2.16.0 軟體更新	21
公有元件更新	21
AWS IoT Greengrass Core v2.15.1 軟體更新	23
公有元件更新	23
AWS IoT Greengrass Core v2.15.0 軟體更新	24
公有元件更新	24
AWS IoT Greengrass Core v2.14.3 軟體更新	26
公有元件更新	26
AWS IoT Greengrass Core v2.14.2 軟體更新	27
公有元件更新	28
AWS IoT Greengrass Core v2.14.1 軟體更新	28
公有元件更新	29
AWS IoT Greengrass Core v2.14.0 軟體更新	30
公有元件更新	31
AWS IoT Greengrass Core v2.13.0 軟體更新	33
公有元件更新	33
AWS IoT Greengrass Core v2.12.6 軟體更新	35
公有元件更新	35
AWS IoT Greengrass Core v2.12.5 軟體更新	36
公有元件更新	36

AWS IoT Greengrass Core v2.12.4 軟體更新	37
公有元件更新	37
AWS IoT Greengrass Core v2.12.3 軟體更新	38
公有元件更新	38
AWS IoT Greengrass Core v2.12.2 軟體更新	40
公有元件更新	40
AWS IoT Greengrass Core v2.12.1 軟體更新	41
公有元件更新	42
AWS IoT Greengrass Core v2.12.0 軟體更新	43
公有元件更新	43
AWS IoT Greengrass Core v2.11.3 軟體更新	44
公有元件更新	44
AWS IoT Greengrass Core v2.11.2 軟體更新	46
公有元件更新	46
AWS IoT Greengrass Core v2.11.1 軟體更新	47
公有元件更新	47
AWS IoT Greengrass Core v2.11.0 軟體更新	48
公有元件更新	48
AWS IoT Greengrass Core v2.10.3 軟體更新	49
公有元件更新	50
AWS IoT Greengrass Core v2.10.2 軟體更新	50
公有元件更新	51
AWS IoT Greengrass Core v2.10.1 軟體更新	52
公有元件更新	52
AWS IoT Greengrass Core v2.10.0 軟體更新	53
公有元件更新	54
AWS IoT Greengrass Core v2.9.6 軟體更新	55
公有元件更新	55
AWS IoT Greengrass Core v2.9.5 軟體更新	56
公有元件更新	56
AWS IoT Greengrass Core v2.9.4 軟體更新	57
公有元件更新	57
AWS IoT Greengrass Core v2.9.3 軟體更新	58
公有元件更新	58
AWS IoT Greengrass Core v2.9.2 軟體更新	59
公有元件更新	59

AWS IoT Greengrass Core v2.9.1 軟體更新	60
公有元件更新	61
AWS IoT Greengrass Core v2.9.0 軟體更新	62
公有元件更新	62
AWS IoT Greengrass Core v2.8.1 軟體更新	64
公有元件更新	64
AWS IoT Greengrass Core v2.8.0 軟體更新	65
公有元件更新	65
AWS IoT Greengrass Core v2.7.0 軟體更新	67
公有元件更新	67
AWS IoT Greengrass Core v2.6.0 軟體更新	69
公有元件更新	69
AWS IoT Greengrass Core v2.5.6 軟體更新	72
公有元件更新	72
AWS IoT Greengrass Core v2.5.5 軟體更新	73
公有元件更新	74
AWS IoT Greengrass Core v2.5.4 軟體更新	75
公有元件更新	75
AWS IoT Greengrass Core v2.5.3 軟體更新	76
公有元件更新	76
AWS IoT Greengrass Core v2.5.2 軟體更新	77
公有元件更新	77
AWS IoT Greengrass Core v2.5.1 軟體更新	78
公有元件更新	78
AWS IoT Greengrass Core v2.5.0 軟體更新	79
平台支援更新	80
公有元件更新	80
AWS IoT Greengrass Core v2.4.0 軟體更新	83
公有元件更新	84
AWS IoT Greengrass Core v2.3.0 軟體更新	86
公有元件更新	86
AWS IoT Greengrass Core v2.2.0 軟體更新	87
公有元件更新	88
AWS IoT Greengrass Core v2.1.0 軟體更新	90
平台支援更新	91
公有元件更新	91

AWS IoT Greengrass Core v2.0.5 軟體更新	97
公有元件更新	97
AWS IoT Greengrass Core v2.0.4 軟體更新	98
公有元件更新	98
從第 1 版遷移	100
遷移概觀	100
V1 和 V2 之間的差異	101
選擇您的執行時間	109
事件來源相容性矩陣	109
執行時間選擇決策流程	110
後續步驟	110
設定測試裝置	111
設定測試裝置 (Greengrass 核)	111
設定測試裝置 (Greengrass nucleus lite)	116
將 V1 核心裝置升級至 V2	184
步驟 1：安裝 AWS IoT Greengrass Core 軟體 v2.x	184
步驟 2：將 Greengrass V2 元件部署至核心裝置	187
開始使用	189
先決條件	190
步驟 1：設定 AWS 帳戶	191
註冊 AWS 帳戶	191
建立具有管理存取權的使用者	192
步驟 2：設定您的環境	193
步驟 3：安裝 AWS IoT Greengrass 核心軟體	198
安裝 AWS IoT Greengrass Core 軟體（主控台）	199
安裝 AWS IoT Greengrass 核心軟體 (CLI)	204
執行 Greengrass 軟體 (Linux)	209
在裝置上驗證 Greengrass CLI 安裝	210
步驟 4：在您的裝置上開發和測試元件	212
步驟 5：在 AWS IoT Greengrass 服務中建立元件	222
步驟 6：部署您的元件	233
後續步驟	238
設定 Greengrass 核心裝置	239
支援平台	239
裝置要求	239
Lambda 函數需求	240

設定 AWS 帳戶	241
安裝 AWS IoT Greengrass Core 軟體	242
使用自動佈建安裝	244
使用手動佈建安裝	258
使用機群佈建安裝	294
使用自訂佈建安裝	335
安裝程式引數	351
執行 AWS IoT Greengrass 核心軟體	355
檢查 AWS IoT Greengrass Core 軟體是否以系統服務的形式執行	355
執行 AWS IoT Greengrass Core 軟體做為系統服務	357
在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體	358
在 Docker AWS IoT Greengrass 中執行	358
支援平台和需求	359
軟體下載	360
選擇如何佈建 AWS 資源	360
從 Dockerfile 建置 AWS IoT Greengrass 映像	360
使用自動佈建在 Docker AWS IoT Greengrass 中執行	365
使用手動佈建在 Docker AWS IoT Greengrass 中執行	372
在 Docker AWS IoT Greengrass 容器中進行故障診斷	389
設定 AWS IoT Greengrass 核心軟體	392
部署 Greengrass 核元件	393
將 Greengrass 核設定為系統服務	393
使用 JVM 選項控制記憶體配置	396
設定執行元件的使用者	398
設定系統資源限制	402
連線至連接埠 443 或透過網路代理	405
使用私有 CA 簽署的裝置憑證	412
設定 MQTT 逾時和快取設定	412
在 IPv6 網路上設定 Greengrass Nucleus	412
更新 AWS IoT Greengrass 核心軟體 (OTA)	413
要求	413
核心裝置的考量事項	414
Greengrass 核更新行為	414
執行 OTA 更新	415
解除安裝 AWS IoT Greengrass Core 軟體	415
教學	419

教學課程影片	419
影片	419
開發延遲元件更新的元件	420
先決條件	421
步驟 1：安裝 Greengrass 開發套件 CLI	422
步驟 2：開發延遲更新的元件	423
步驟 3：將元件發佈至 AWS IoT Greengrass 服務	431
步驟 4：在核心裝置上部署和測試元件	434
透過 MQTT 與本機 IoT 裝置互動	438
先決條件	439
步驟 1：檢閱和更新核心裝置 AWS IoT 政策	440
步驟 2：啟用用戶端裝置支援	441
步驟 3：連接用戶端裝置	446
步驟 4：開發與用戶端裝置通訊的元件	449
步驟 5：開發與用戶端裝置陰影互動的元件	455
使用 TPM 保護 Greengrass Nucleus	480
先決條件	480
步驟 1：安裝 TPM2 工具和相依性	481
步驟 2：初始化 PKCS#11 存放區並建立槽	482
步驟 3：建立權杖和金鑰	483
步驟 4：產生憑證簽署請求 (CSR)	485
步驟 5：建立物件憑證	486
步驟 6：將物件憑證匯入 TPM	486
步驟 7：擷取 PKCS#11 物件 URL	486
步驟 8：使用 TPM2 支援設定和安裝 Greengrass	488
步驟 9：驗證安裝	489
疑難排解	489
後續步驟	490
使用 TPM 保護 Greengrass Nucleus Lite	490
先決條件	491
步驟 1：設定 NitroTPM 執行個體	491
步驟 2：安裝和設定 TPM 工具	492
步驟 3：設定 OpenSSL TPM2 供應商	492
步驟 4：產生持久性 TPM 金鑰	493
步驟 5：產生憑證簽署請求 (CSR)	493
步驟 6：從 CSR 建立憑證	494

步驟 7：使用 TPM 支援設定 Greengrass Nucleus Lite	495
疑難排解	495
後續步驟	496
開始使用 SageMaker AI Edge Manager	496
先決條件	497
在 SageMaker AI Edge Manager 中設定	499
建立範例元件	500
執行範例影像分類推論	501
執行範例影像分類推論	505
先決條件	506
步驟 1：訂閱預設通知主題	506
步驟 2：部署 TensorFlow Lite 影像分類元件	507
步驟 3：檢視推論結果	508
後續步驟	510
從攝影機對影像執行範例影像分類推論	511
先決條件	511
步驟 1：在您的裝置上設定攝影機模組	512
步驟 2：驗證預設通知主題的訂閱	514
步驟 3：修改 TensorFlow Lite 影像分類元件組態並進行部署	515
步驟 4：檢視推論結果	517
後續步驟	517
元件	518
AWS提供的元件	518
Greengrass 核	527
Greengrass nucleus lite	563
用戶端裝置身分驗證	569
CloudWatch 指標	636
AWS IoT Device Defender	658
磁碟多工緩衝處理	674
Docker 應用程式管理員	678
Kinesis Video Streams 的邊緣連接器	688
Greengrass CLI	695
IP 偵測器	708
Firehose	718
Lambda 啟動器	735
Lambda 管理員	738

Lambda 執行期	747
傳統訂閱路由器	749
本機偵錯主控台	760
日誌管理員	776
機器學習元件	819
Modbus-RTU 通訊協定轉接器	924
MQTT 橋接器	955
MQTT 3.1.1 代理程式 (Moquette)	976
MQTT 5 代理程式 (EMQX)	983
Nucleus 遙測發射器	999
PKCS#11 供應商	1011
秘密管理員	1019
安全通道	1030
Shadow 管理員	1040
Amazon SNS	1067
串流管理員	1083
系統日誌轉寄站	1096
Systems Manager 代理程式	1101
字符交換服務	1107
IoT SiteWise OPC UA 收集器	1111
IoT SiteWise OPC UA 資料來源模擬器	1119
IoT SiteWise 發佈者	1122
IoT SiteWise 處理器	1135
發行者支援的元件	1148
AIShield.Edge	1149
AI EdgeLabs 感應器	1149
Greengrass S3 擷取器	1150
社群元件	1150
Greengrass 開發工具	1154
Greengrass 開發套件 CLI	1154
Greengrass 命令列界面	1181
使用 Greengrass 測試架構	1198
開發元件	1211
元件生命週期	1213
元件類型	1213
建立元件	1214

使用本機部署測試元件	1225
發佈元件以部署	1228
與 AWS 服務互動	1233
執行 Docker 容器	1236
配方參考	1258
環境變數	1285
將元件部署至裝置	1286
核心裝置部署	1286
平台相依性解析	1286
元件相依性解析	1286
從物件群組移除裝置	1293
部署	1294
部署選項	1294
建立部署	1296
建立子部署	1313
修訂部署	1316
取消部署	1318
檢查部署狀態	1319
日誌記錄和監控	1323
監控工具	1323
監控 Greengrass 日誌	1324
存取檔案系統日誌	1324
存取 CloudWatch Logs	1326
存取系統服務日誌	1328
啟用記錄至 CloudWatch Logs	1330
設定 的記錄 AWS IoT Greengrass	1331
AWS CloudTrail 日誌	1332
使用 CloudTrail 記錄 API 呼叫	1333
AWS IoT Greengrass V2 CloudTrail 中的資訊	1333
AWS IoT Greengrass CloudTrail 中的資料事件	1334
AWS IoT Greengrass CloudTrail 中的管理事件	1338
了解 AWS IoT Greengrass V2 日誌檔案項目	1338
收集系統運作狀態遙測資料	1340
遙測指標	1341
設定遙測代理程式設定	1344
訂閱 EventBridge 中的遙測資料	1345

取得部署和元件運作狀態通知	1353
部署狀態變更事件	1354
元件狀態變更事件	1355
建立 EventBridge 規則的先決條件	1357
設定裝置運作狀態通知（主控台）	1358
設定裝置運作狀態通知 (CLI)	1359
設定裝置運作狀態通知 (CloudFormation)	1360
另請參閱	1360
檢查核心裝置狀態	1360
檢查核心裝置的運作狀態	1361
檢查核心裝置群組的運作狀態	1361
檢查核心裝置元件狀態	1362
執行 Lambda 函數	1363
要求	1363
設定 Lambda 函數生命週期	1364
設定 Lambda 函數容器化	1365
將 Lambda 函數匯入為元件（主控台）	1367
步驟 1：選擇要匯入的 Lambda 函數	1367
步驟 2：設定 Lambda 函數參數	1368
步驟 3：（選用）指定 Lambda 函數支援的平台	1370
步驟 4：（選用）指定 Lambda 函數的元件相依性	1370
步驟 5：（選用）在容器中執行 Lambda 函數	1371
步驟 6：建立 Lambda 函數元件	1372
將 Lambda 函數匯入為元件 (CLI)	1372
步驟 1：定義 Lambda 函數組態	1373
步驟 2：建立 Lambda 函數元件	1392
與 Greengrass 核、其他元件和 通訊 AWS IoT Core	1394
IPC 用戶端版本	1395
支援的 SDK	1395
連線至 AWS IoT Greengrass Core IPC 服務	1396
授權元件執行 IPC 操作	1402
授權政策中的萬用字元	1403
授權政策中的配方變數	1403
授權政策中的特殊字元	1404
授權政策範例	1404
訂閱 IPC 事件串流	1408

定義訂閱處理常式	1408
範例訂閱處理常式	1410
IPC 最佳實務	1422
發佈/訂閱本機訊息	1424
最低 SDK 版本	1424
Authorization	1425
PublishToTopic	1426
SubscribeToTopic	1437
範例	1452
發佈/訂閱 AWS IoT Core MQTT 訊息	1486
最低 SDK 版本	1487
Authorization	1487
PublishToIoTCore	1491
SubscribeToIoTCore	1502
範例	1520
與元件生命週期互動	1539
最低 SDK 版本	1540
Authorization	1540
UpdateState	1541
SubscribeToComponentUpdates	1544
DeferComponentUpdate	1545
PauseComponent	1546
ResumeComponent	1548
與元件組態互動	1549
最低 SDK 版本	1549
GetConfiguration	1550
UpdateConfiguration	1553
SubscribeToConfigurationUpdate	1556
SubscribeToValidateConfigurationUpdates	1561
SendConfigurationValidityReport	1562
擷取秘密值	1563
最低 SDK 版本	1563
Authorization	1563
GetSecretValue	1565
範例	1570
與本機陰影互動	1576

最低 SDK 版本	1577
Authorization	1577
GetThingShadow	1588
UpdateThingShadow	1595
DeleteThingShadow	1603
ListNamedShadowsForThing	1609
管理本機部署和元件	1616
最低 SDK 版本	1617
Authorization	1617
CreateLocalDeployment	1619
ListLocalDeployments	1622
GetLocalDeploymentStatus	1623
ListComponents	1624
GetComponentDetails	1625
RestartComponent	1626
StopComponent	1629
CreateDebugPassword	1630
驗證和授權用戶端裝置	1630
最低 SDK 版本	1631
Authorization	1632
VerifyClientDeviceIdentity	1633
GetClientDeviceAuthToken	1634
AuthorizeClientDeviceAction	1635
SubscribeToCertificateUpdates	1635
與本機 IoT 裝置互動	1638
用戶端裝置元件	1638
將用戶端裝置連接至核心裝置	1640
需求	1641
用戶端裝置支援的 Greengrass 元件	1653
設定雲端探索（主控台）	1655
設定雲端探索 (AWS CLI)	1655
關聯用戶端裝置	1655
離線時驗證用戶端	1657
管理核心裝置端點	1658
選擇 MQTT 代理程式	1665
連線至 MQTT 代理程式	1666

測試通訊	1668
Greengrass 探索 RESTful API	1678
在用戶端裝置與 之間轉送 MQTT 訊息 AWS IoT Core	1684
設定和部署 MQTT 橋接器元件	1685
轉送 MQTT 訊息	1686
與元件中的用戶端裝置互動	1687
設定和部署 MQTT 橋接器元件	1687
從用戶端裝置接收 MQTT 訊息	1688
將 MQTT 訊息傳送至用戶端裝置	1689
與用戶端裝置影子互動和同步	1689
先決條件	1690
讓影子管理員與用戶端裝置通訊	1690
與元件中的用戶端裝置陰影互動	1693
使用 同步用戶端裝置影子 AWS IoT Core	1693
將 IPv6 用於本機傳訊	1693
設定 IP 偵測器以使用 IPv6	1693
疑難排解	1697
Greengrass 探索問題	1697
MQTT 連線問題	1704
與裝置陰影互動	1710
與元件中的陰影互動	1710
擷取和修改陰影狀態	1711
對陰影狀態變更做出反應	1711
使用 同步本機裝置陰影 AWS IoT Core	1712
先決條件	1713
設定陰影管理員元件	1713
同步本機陰影	1715
陰影合併衝突行為	1715
管理資料串流	1716
串流管理工作流程	1716
要求	1717
資料安全	1718
本機資料安全性	1718
用戶端身分驗證	1718
另請參閱	1719
設定串流管理員	1719

串流管理員參數	1719
另請參閱	1721
建立使用串流管理員的自訂元件	1722
定義使用串流管理員的元件配方	1722
連線至應用程式程式碼中的串流管理員	1734
使用 StreamManagerClient 搭配串流	1736
建立訊息串流	1737
附加訊息	1741
讀取訊息	1747
列出串流	1750
描述訊息串流	1751
更新訊息串流	1753
刪除訊息串流	1757
另請參閱	1758
支援雲端目的地的匯出組態	1759
執行機器學習推論	1774
AWS IoT Greengrass ML 推論的運作方式	1774
第 2 AWS IoT Greengrass 版有何不同？	1775
要求	1775
支援的模型來源	1776
支援的執行期	1776
機器學習元件	1776
使用 SageMaker AI Edge Manager	1780
運作方式	1781
要求	1782
開始使用 SageMaker AI Edge Manager	1783
自訂您的機器學習元件	1783
修改公有推論元件的組態	1784
搭配範例推論元件使用自訂模型	1786
建立自訂機器學習元件	1789
建立自訂推論元件	1791
疑難排解	1798
無法擷取程式庫	1799
Cannot open shared object file	1799
Error: ModuleNotFoundError: No module named '<library>'	1800
未偵測到支援 CUDA 的裝置	1801

沒有此類檔案或目錄	1801
RuntimeError: module compiled against API version 0xf but this version of NumPy is	
<version>	1802
picamera.exc.PiCameraError: Camera is not enabled	1802
記憶體錯誤	1803
磁碟空間錯誤	1803
逾時錯誤	1803
使用 管理核心裝置 AWS Systems Manager	1804
安裝 Systems Manager Agent	1805
步驟 1：完成一般 Systems Manager 設定步驟	1805
步驟 2：建立 Systems Manager 的 IAM 服務角色	1805
步驟 3：將許可新增至權杖交換角色	1806
步驟 4：部署 Systems Manager Agent 元件	1810
步驟 5：使用 Systems Manager 驗證核心裝置註冊	1812
解除安裝 Systems Manager 代理程式	1813
步驟 1：從 Systems Manager 取消註冊核心裝置	1814
步驟 2：解除安裝 Systems Manager Agent 元件	1814
步驟 3：解除安裝 Systems Manager Agent 軟體	1815
安全	1816
資料保護	1817
資料加密	1818
硬體安全整合	1819
裝置身分驗證和授權	1830
X.509 憑證	1830
AWS IoT 政策	1831
更新核心裝置的 AWS IoT 政策	1835
最低 AWS IoT 政策	1840
支援用戶端裝置的最低 AWS IoT 政策	1843
用戶端裝置的最低 AWS IoT 政策	1844
身分與存取管理	1846
目標對象	1847
使用身分驗證	1847
使用政策管理存取權	1848
另請參閱	1849
AWS IoT Greengrass 如何使用 IAM	1849
身分型政策範例	1853

授權核心裝置與 AWS 服務互動	1855
安裝程式佈建資源的最低 IAM 政策	1860
Greengrass 服務角色	1864
AWS 受管政策	1872
預防跨服務混淆代理人	1878
針對識別和存取問題進行故障診斷	1878
允許透過代理或防火牆的裝置流量	1880
基本操作的端點	1880
使用自動佈建進行安裝的端點	1883
AWS所提供元件的端點	1884
法規遵循驗證	1884
FIPS 端點	1885
使用部署啟用 FIPS 端點	1885
使用手動資源佈建搭配 FIPS 端點安裝 Nucleus	1887
使用機群佈建安裝 FIPS 端點	1918
使用自動資源佈建安裝 FIPS 端點	1934
FIPS 合規第一方元件	1949
恢復能力	1950
基礎設施安全性	1951
組態與漏洞分析	1951
程式碼完整性	1952
VPC 端點 (AWS PrivateLink)	1953
AWS IoT Greengrass VPC 端點的考量事項	1953
建立 AWS IoT Greengrass 控制平面操作的介面 VPC 端點	1954
為 建立 VPC 端點政策 AWS IoT Greengrass	1954
在 VPC 中操作 AWS IoT Greengrass 核心裝置	1955
安全最佳實務	1959
盡可能授予最低的許可	1959
請勿在 Greengrass 元件中硬式編碼登入資料	1959
請勿記錄敏感資訊	1959
讓裝置的時鐘保持同步	1960
密碼套件建議	1960
另請參閱	1960
使用 AWS IoT Device Tester for AWS IoT Greengrass V2	1961
AWS IoT Greengrass 資格套件	1961
自訂測試套件	1962

支援的版本	1962
最新版 IDT for AWS IoT Greengrass V2	1962
的早期 IDT 版本 AWS IoT Greengrass	1963
不支援的 AWS IoT Device Tester for AWS IoT Greengrass V2 版本	1964
下載 IDT for AWS IoT Greengrass V2	1968
手動下載 IDT	1969
以程式設計方式下載 IDT	1970
使用 IDT AWS IoT Greengrass 執行資格套件	1975
測試套件版本	1975
測試群組描述	1976
先決條件	1979
設定您的裝置以執行 IDT 測試	1999
設定 IDT 設定	2008
執行 AWS IoT Greengrass 資格套件	2024
了解結果和日誌	2027
使用 IDT 來開發和執行您自己的測試套件	2030
下載最新版本的 IDT for AWS IoT Greengrass	1979
測試套件建立工作流程	2031
教學課程：建置並執行範例 IDT 測試套件	2031
教學課程：開發簡單的 IDT 測試套件	2037
建立 IDT 測試套件組態檔案	2046
設定 IDT 測試協調器	2053
設定 IDT 狀態機器	2059
建立 IDT 測試案例可執行檔	2081
使用 IDT 內容	2087
設定測試執行器的設定	2091
偵錯並執行自訂測試套件	2101
檢閱 IDT 測試結果和日誌	2104
IDT 用量指標	2110
對 IDT for AWS IoT Greengrass V2 進行故障診斷	2116
在哪裡尋找錯誤	2116
解決 IDT for AWS IoT Greengrass V2 錯誤	2117
適用於 AWS IoT Device Tester 的支援政策 AWS IoT Greengrass	2123
Greengrass 型 IoT 解決方案	2124
Eurotech	2124
疑難排解	2125

檢視 AWS IoT Greengrass 核心軟體和元件日誌	2125
AWS IoT Greengrass 核心軟體問題	2125
ListDeployments API 中的 ThrottlingException	2127
無法設定核心裝置	2127
無法啟動 AWS IoT Greengrass Core 軟體做為系統服務	2127
無法將 核設定為系統服務	2127
無法連線至 AWS IoT Core	2128
記憶體不足錯誤	2128
無法安裝 Greengrass CLI	2128
User root is not allowed to execute	2128
com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/ group to run with	2129
Failed to map segment from shared object: operation not permitted	2129
無法設定 Windows 服務	2129
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager	2130
com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime	2130
software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid	2131
software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy	2132
Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request	2132
Operation aws.greengrass#<operation> is not supported by Greengrass	2133
java.io.FileNotFoundException: <stream-manager-store-root-dir>/ stream_manager_metadata_store (Permission denied)	2133
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist	2133
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn> .	2134
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed	2135
java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi	2135
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED	2135
Greengrass core device stuck on nucleus v2.12.3	2135

Greengrass nucleus v2.14.0 systemd template issue	2138
AWS IoT Greengrass 雲端問題	2138
An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null	2138
Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}	2139
INACTIVE deployment status	2139
核心裝置部署問題	2139
Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact	2140
Error:	
com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.	2141
Error:	
com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>	2142
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility	2142
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	2143
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	2143
Info:	
com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration	2145
Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy	2145
Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration	2145

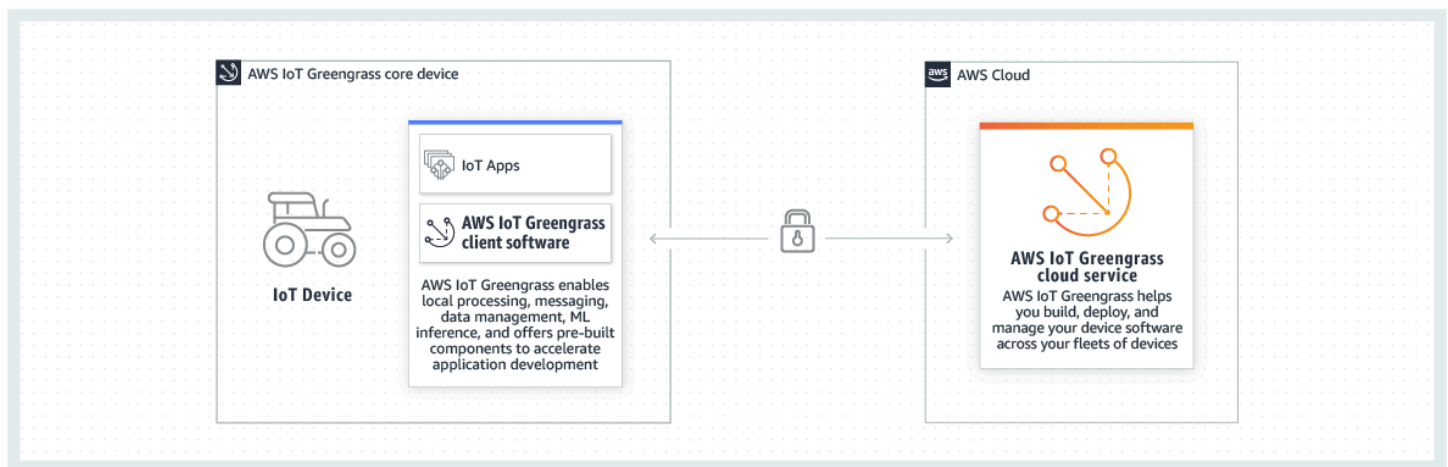
Caused by:	
software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException:	
null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)	2146
核心裝置元件問題	2146
Warn: '<command>' is not recognized as an internal or external command	2147
Python 指令碼不會記錄訊息	2147
變更預設組態時，元件組態不會更新	2148
awsiot.greengrasscoreipc.model.UnauthorizedError	2149
com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"	2150
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)	2150
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)	2152
com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers	2152
Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"	2152
copyFrom: <configurationPath> is already a container, not a leaf	2153
com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'	2153
java.io.IOException: Cannot run program "cmd": [LogonUser] The password for this account has expired.	2154
aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant	2155
核心裝置 Lambda 函數元件問題	2155
The following cgroup subsystems are not mounted: devices, memory	2156
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>'	2156
元件版本已停止	2156
Greengrass CLI 問題	2157
java.lang.RuntimeException: Unable to create ipc client	2157
AWS CLI 問題	2157
Error: Invalid choice: 'greengrassv2'	2158
詳細部署錯誤代碼	2158
許可錯誤	2159

請求錯誤	2161
元件配方錯誤	2163
AWS 元件錯誤、使用者元件錯誤、元件錯誤	2164
裝置錯誤	2165
相依性錯誤	2166
HTTP 錯誤	2167
網路錯誤	2168
Nucleus 錯誤	2168
伺服器錯誤	2169
雲端服務錯誤	2169
一般錯誤	2170
未知錯誤	2171
詳細的元件狀態碼	2171
標記您的 資源	2174
在 中使用標籤 AWS IoT Greengrass V2	2174
使用 標記 AWS 管理主控台	2174
使用 AWS IoT Greengrass V2 API 標記	2174
搭配 IAM 政策使用標籤	2175
AWS CloudFormation 資源	2177
AWS IoT Greengrass 和 CloudFormation 範本	2177
ComponentVersion 範本範例	2177
部署範本範例	2178
進一步了解 CloudFormation	2179
開放原始碼軟體	2180
文件歷史紀錄	2181
AWS 詞彙表	2223
.....	mmccxxiv

什麼是 AWS IoT Greengrass ？

AWS IoT Greengrass 是一種開放原始碼物聯網 (IoT) 節點執行期和雲端服務，可協助您在裝置上建置、部署和管理 IoT 應用程式。您可以使用 AWS IoT Greengrass 建置軟體，讓裝置在本機對產生的資料採取行動、根據機器學習模型執行預測，以及篩選和彙總裝置資料。AWS IoT Greengrass 可讓裝置收集和分析更接近產生資料的位置的資料、自動回應本機事件，以及安全地與本機網路上的其他裝置通訊。Greengrass 裝置也可以安全地與通訊，AWS IoT Core 並將 IoT 資料匯出至 AWS 雲端。您可以使用 AWS IoT Greengrass，使用稱為元件的預先建置軟體模組來建置邊緣應用程式，這些模組可將邊緣裝置連線至 AWS 服務或第三方服務。您也可以使用 AWS IoT Greengrass，使用 Lambda 函數、Docker 容器、原生作業系統程序或您選擇的自訂執行時間來封裝和執行軟體。

下列範例顯示 AWS IoT Greengrass 裝置如何與互動 AWS 雲端。



新功能

AWS IoT Greengrass V2 推出新功能和改進功能。以下包含有關第 2 版中提供的新功能的詳細資訊。

- [中的新功能 AWS IoT Greengrass Version 2](#)

對於 的第一次使用者 AWS IoT Greengrass

如果您是新手 AWS IoT Greengrass，建議您檢閱以下章節：

- [AWS IoT Greengrass 運作方式](#)

接下來，遵循[入門教學](#)課程來嘗試的基本功能 AWS IoT Greengrass。在本教學課程中，您會在裝置上安裝 AWS IoT Greengrass Core 軟體、開發 Hello World 元件，以及封裝該元件以進行部署。

對於的現有使用者 AWS IoT Greengrass V1

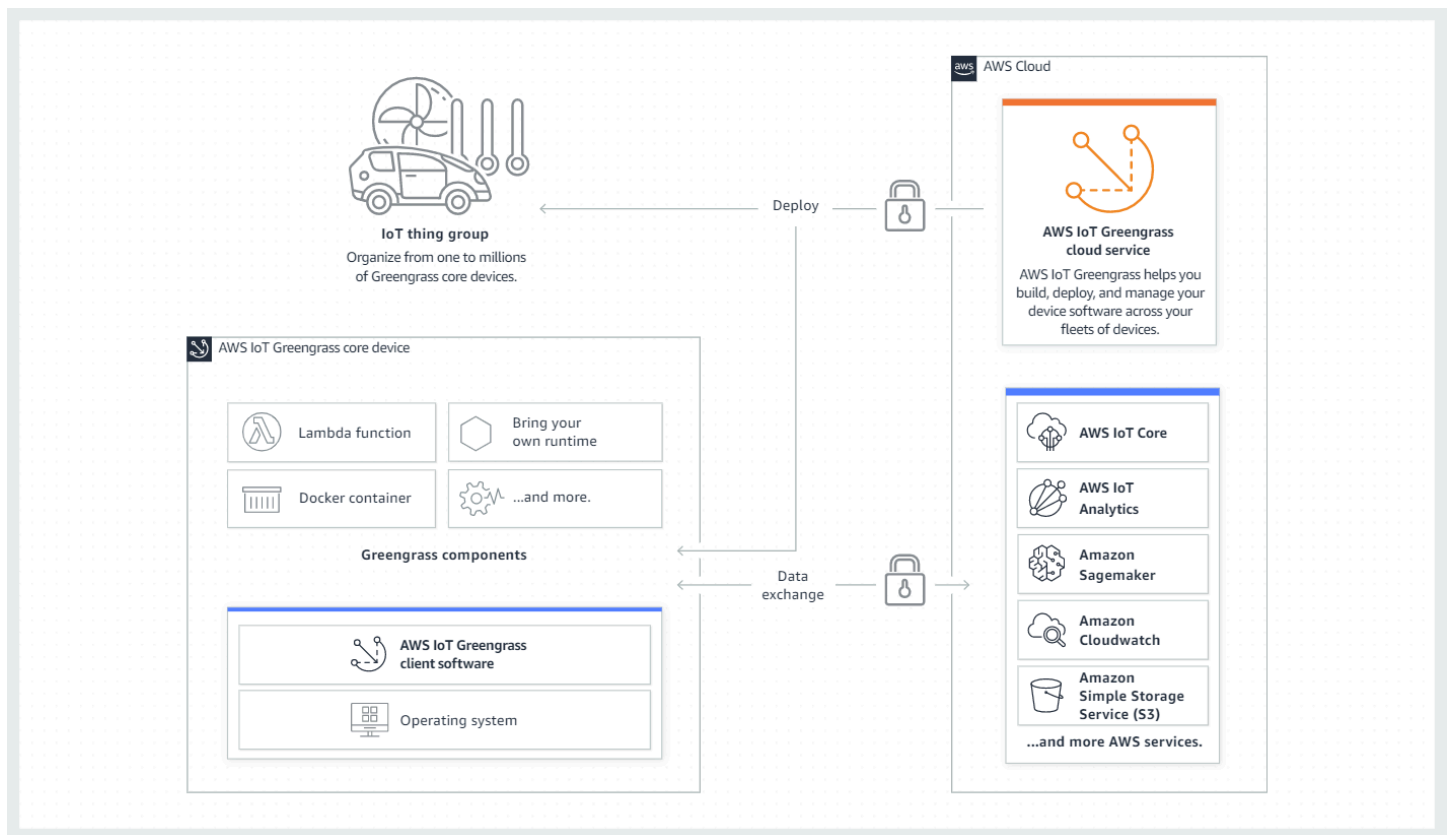
終止支援通知：將於 2026 AWS 年 10 月 7 日終止對的支援 AWS IoT Greengrass Version 1。2026 年 10 月 7 日之後，您將無法再存取 AWS IoT Greengrass V1 主控台或 AWS IoT Greengrass V1 資源。如需如何從第 1 版移至第 2 版的詳細資訊，請參閱 [從第 1 AWS IoT Greengrass 版遷移](#)。

AWS IoT Greengrass 運作方式

對於具有 ARM 或 x86 架構的裝置，AWS IoT Greengrass 用戶端軟體也稱為 AWS IoT Greengrass Core 軟體，在 Windows 和 Linux 發行版本上執行，例如 Ubuntu 或 Raspberry Pi OS。使用 AWS IoT Greengrass，您可以對裝置進行程式設計，以根據其產生的資料在本機採取行動、根據機器學習模型執行預測，以及篩選和彙總裝置資料。AWS IoT Greengrass 啟用 AWS Lambda 函式、Docker 容器、原生作業系統程序或您選擇的自訂執行時間的本機執行。

AWS IoT Greengrass 提供稱為元件的預先建置軟體模組，可讓您輕鬆擴展邊緣裝置功能。AWS IoT Greengrass 元件可讓您連線至邊緣 AWS 的服務和第三方應用程式。在您開發 IoT 應用程式之後，AWS IoT Greengrass 可讓您在欄位中遠端部署、設定和管理裝置機群上的這些應用程式。

下列範例顯示 AWS IoT Greengrass 裝置如何與中的 AWS IoT Greengrass 雲端服務和其他 AWS 服務互動 AWS 雲端。



的主要概念 AWS IoT Greengrass

以下是了解和使用的基本概念 AWS IoT Greengrass：

AWS IoT 物件

AWS IoT 物件是特定裝置或邏輯實體的表示法。有關物件的資訊會存放在 AWS IoT 登錄檔中。

Greengrass 核心裝置

執行 AWS IoT Greengrass Core 軟體的裝置。Greengrass 核心裝置是 AWS IoT 物件。您可以將多個核心裝置新增至 AWS IoT 物件群組，以建立和管理 Greengrass 核心裝置的群組。如需詳細資訊，請參閱[設定 AWS IoT Greengrass 核心裝置](#)。

Greengrass 用戶端裝置

透過 MQTT 連線至 Greengrass 核心裝置並與之通訊的裝置。Greengrass 用戶端裝置是 AWS IoT 實物。核心裝置可以處理、篩選和彙總連接到該核心裝置之用戶端裝置的資料。您可以設定核心裝置，在用戶端裝置、AWS IoT Core 雲端服務和 Greengrass 元件之間轉送 MQTT 訊息。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

用戶端裝置可以執行 [FreeRTOS](#)，或使用 [AWS IoT Device SDK](#) 或 [Greengrass 探索 API](#) 來取得其可連線之核心裝置的相關資訊。

Greengrass 元件

部署至 Greengrass 核心裝置並在其上執行的軟體模組。使用開發和部署的所有軟體 AWS IoT Greengrass 都會建模為元件。AWS IoT Greengrass 提供預先建置的公有元件，提供您可以在應用程式中使用的功能。您也可以在本機裝置或雲端開發自己的自訂元件。開發自訂元件之後，您可以使用 AWS IoT Greengrass 雲端服務將其部署到單一或多個核心裝置。您可以建立自訂元件，並將該元件部署至核心裝置。當您這麼做時，核心裝置會下載下列資源來執行元件：

- 配方：透過定義元件詳細資訊、組態和參數來描述軟體模組的 JSON 或 YAML 檔案。
- 成品：來源碼、二進位檔或指令碼，定義將在您的裝置上執行的軟體。您可以從頭開始建立成品，也可以使用 Lambda 函數、Docker 容器或自訂執行時間建立元件。
- 相依性：元件之間的關係，可讓您強制執行相依元件的自動更新或重新啟動。例如，您可以擁有依賴加密元件的安全訊息處理元件。這可確保加密元件的任何更新都會自動更新並重新啟動訊息處理元件。

如需詳細資訊，請參閱[AWS提供的元件](#)及[開發 AWS IoT Greengrass 元件](#)。

部署

傳送元件並將所需元件組態套用至目的地目標裝置的程序，可以是單一 Greengrass 核心裝置或一組 Greengrass 核心裝置。部署會自動將任何更新的元件組態套用至目標，並包含定義為相依性的任何其他元件。您也可以複製現有的部署，以建立使用相同元件但部署到不同目標的新部署。部署是連續的，這表示您對元件所做的任何更新或部署的元件組態都會自動傳送至所有目的地目標。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

AWS IoT Greengrass 核心軟體

從 2.14 版開始，AWS IoT Greengrass 提供其裝置執行期的兩種替代實作，稱為核的可執行檔。第一個和先前唯一的核是在 Java 中實作。此選項可在架構和作業系統之間提供最大的可攜性。不過，它也隨附對 Java 虛擬機器的相依性，導致大量的記憶體使用量。

第二個新增的核是在 C 中實作。此選項可大幅減少其使用量。不過，它需要針對不同的目標架構和作業系統分別進行分佈（或從來源編譯）。當需要區分兩者時，我們會將第一個實作稱為 Greengrass 核，後者稱為 Greengrass 核精簡。

- 選用元件：這些可設定的元件由提供，AWS IoT Greengrass 並在您的邊緣裝置上啟用其他功能。根據您的需求，您可以選擇要部署到裝置的選用元件，例如資料串流、本機機器學習推論或本機命令列界面。如需詳細資訊，請參閱[AWS提供的元件](#)。

您可以將新版本的元件部署到裝置，以升級您的 AWS IoT Greengrass Core 軟體。

的功能 AWS IoT Greengrass

AWS IoT Greengrass Version 2 包含下列元素：

- 軟體分佈
 - [Greengrass 核心元件](#)，這是 AWS IoT Greengrass Core 軟體的最低安裝量。此元件可管理 Greengrass 元件的部署、協同運作和生命週期管理。
 - 其他選用 [AWS 提供的元件](#)，可與 服務、通訊協定和軟體整合。
 - [Greengrass 開發工具](#)，可用來建立、測試、建置、發佈和部署自訂 Greengrass 元件。
 - AWS IoT Device SDK 包含自訂 Greengrass 元件的 [處理程序間通訊 \(IPC\) 程式庫](#)，以及用戶端裝置的 [Greengrass 探索程式庫](#)。
 - 串流管理員 SDK，可用來[管理核心裝置上的資料串流](#)。
- 雲端服務
 - AWS IoT Greengrass V2 API
 - AWS IoT Greengrass V2 主控台

AWS IoT Greengrass 核心軟體

您可以使用在邊緣裝置上執行 AWS IoT Greengrass 的核心軟體來執行下列動作：

- 在本機裝置上處理資料串流，並自動匯出至 AWS 雲端。如需詳細資訊，請參閱[管理 Greengrass 核心裝置上的資料串流](#)。
- 支援 AWS IoT 和 元件之間的 MQTT 訊息。如需詳細資訊，請參閱[發佈/訂閱 AWS IoT Core MQTT 訊息](#)。
- 與透過 MQTT 連線和通訊的本機裝置互動。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。
- 支援元件之間的本機發佈和訂閱訊息。如需詳細資訊，請參閱[發佈/訂閱本機訊息](#)。
- 部署和叫用元件和 Lambda 函數。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。
- 管理元件生命週期，例如支援安裝和執行指令碼。如需詳細資訊，請參閱[AWS IoT Greengrass 元件配方參考](#)。
- 執行 AWS IoT Greengrass Core 軟體和自訂元件的安全 over-the-air (OTA) 軟體更新。如需詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)及[將 AWS IoT Greengrass 元件部署至裝置](#)。
- 提供安全、加密的本機秘密儲存，以及由 元件控制的存取。如需詳細資訊，請參閱[秘密管理員](#)。

- 使用裝置身分驗證和授權保護裝置與 AWS 雲端之間的連線。如需詳細資訊，請參閱[裝置身分驗證和授權 AWS IoT Greengrass](#)。

您可以透過 AWS IoT Greengrass APIs 來設定和管理 Greengrass 核心裝置，並在其中建立持續的軟體部署。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

某些功能僅支援特定平台。如需詳細資訊，請參閱[Greengrass 功能相容性](#)。

如需支援平台、需求和下載的詳細資訊，請參閱[設定 AWS IoT Greengrass 核心裝置](#)。

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

Greengrass 功能相容性

AWS IoT Greengrass 支援執行各種作業系統的裝置。某些功能僅支援特定作業系統。使用下表了解每個支援的作業系統可使用哪些功能。如需支援的作業系統、需求以及如何設定 Greengrass 核心裝置的詳細資訊，請參閱[設定 AWS IoT Greengrass 核心裝置](#)。

簡訊

功能	Linux	Windows	Greengrass Nucleus Lite (Linux)
在 AWS IoT 和 元件之間交換 MQTT 訊息	 是	 是	 是 (MQTT5 擴充功能除外)
在元件之間交換本機發佈/訂閱訊息	 是	 是	 是
透過 MQTT 與本機 IoT 裝置互動	 是	 是	 否

功能	Linux	Windows	Greengrass Nucleus Lite (Linux)
使用 Modbus-RTU 元件與本機 Modbus-RTU 裝置互動	 是	 否	 否

安全

功能	Linux	Windows	Greengrass lite (Linux)
使用裝置身分驗證和授權保護連線	 是	 是	 是
從 部署和存取安全、加密的秘密 AWS Secrets Manager	 是	 是	 否
使用硬體安全模組 (HSM) 安全地存放裝置的私有金鑰和憑證	 是	 否	<u>是</u>
使用 稽核核心裝置 AWS IoT Device Defender	 是	 是	 否

功能	Linux	Windows	Greengrass lite (Linux)
使用 AWS 登入資料與 AWS 服務互動	 是	 是	 是

安裝

功能	Linux	Windows	Greengrass lite (Linux)
AWS IoT Greengrass 使用自動佈建安裝	 是	 是	 否
AWS IoT Greengrass 使用手動佈建安裝	 是	 是	 是
AWS IoT Greengrass 使用 AWS IoT 機群佈建安裝	 是	 是	 是
AWS IoT Greengrass 使用自訂佈建安裝	<u>是</u>	<u>是</u>	 是

功能	Linux	Windows	Greengrass lite (Linux)
使用預先建置的 Docker 映像 AWS IoT Greengrass 在 Docker 容器中執行	 是	 否	 否

 Note

AWS IoT Greengrass 可以在已啟用系統的 Docker 容器中安裝和執行。

遠端維護和更新

功能	Linux	Windows	Greengrass lite (Linux)
執行安全的over-the-air(OTA) 軟體更新	 是	 是	 是
使用 管理核心裝置 AWS Systems Manager	 是	 否	 否
使用 AWS IoT 安全通道連線至核心裝置	 是	 否	 是

機器學習

功能	Linux	Windows	Greengrass lite (Linux)
使用 Amazon SageMaker AI Edge Manager 執行機器學習推論	 是	 是	 否
使用 Amazon Lookout for Vision 執行機器學習推論	 是	 否	 否
使用 DLR 執行機器學習推論	 是	 是	 否
使用 TensorFlow 執行機器學習推論	 是	 是	 否

元件功能

功能	Linux	Windows	Greengrass lite (Linux)
部署和叫用 Lambda 函數	 是	 否	 否

功能	Linux	Windows	Greengrass lite (Linux)
在元件中執行 Docker 容器	 是	 是	 是
使用串流管理員處理和匯出大量資料串流	 是	 是	 是
使用生命週期指令碼管理元件生命週期	 是	 是	 是
與裝置陰影互動	 是	 是	 否
將日誌上傳至 Amazon CloudWatch Logs	 是	 是	 是
使用 Amazon CloudWatch CloudWatch 指標	 是	 是	 是
使用 Amazon SNS 元件將訊息發佈至 Amazon Simple Notification Service	 是	 否	 否

功能	Linux	Windows	Greengrass lite (Linux)
使用串流管理員將資料發佈至 Amazon Kinesis Data Streams 交付串流	 是	 是	 是
使用 Firehose 元件將資料發佈至 Amazon Data Firehose 交付串流	 是	 否	 否
收集即時系統遙測指標並採取行動	 是	 是	 否
設定元件程序的系統資源限制	 是	 否	 否
暫停和繼續元件程序	 是	 否	 否
AWS IoT SiteWise 使用 AWS IoT SiteWise 元件與整合	 是	 是	 否

功能	Linux	Windows	Greengrass lite (Linux)
使用 Kinesis Video Streams 元件的邊緣連接器，將影片串流發佈至 Amazon Kinesis Video Streams	 是	 否	 否

元件開發

功能	Linux	Windows	Greengrass lite (Linux)
在核心裝置上於本機開發元件	 是	 是	 是
使用 CLI AWS IoT Greengrass 與核心裝置互動	 是	 是	 否
使用本機偵錯主控台與核心裝置互動	 是	 是	 否
在自訂元件中使用 AWS IoT Device SDK for Python	 是	 是	 是

功能	Linux	Windows	Greengrass lite (Linux)
在自訂元件中使用 AWS IoT Device SDK 適用於 C++ 的	 是	 是	 是
在自訂元件中使用 AWS IoT Device SDK 適用於 Java 的	 是	 是	 是

裝置認證

功能	Linux	Windows	Greengrass lite (Linux)
使用 AWS IoT Device Tester AWS IoT Greengrass V2 來驗證 IoT 裝置	 是	 是	 否

選擇您的 nucleus AWS IoT Greengrass 執行時間

自 2.14.0 版起，AWS IoT Greengrass 提供其裝置執行期的兩種替代實作，即稱為 核的可執行檔。雖然兩者的實作差異，但兩個執行時間都與服務 AWS IoT Greengrass 和 APIs 相容，並可讓您使用 Greengrass SDK 部署 提供的元件，AWS 或使用 開發自訂元件。您也可以視需要在相同機群中使用任一類型的核混合裝置。

不過，為了實現所需的可攜性或特定的記憶體節省優勢，請務必確保您在 Greengrass 裝置上部署的核與您打算用來加速解決方案開發的 AWS IoT 元件相容。若要進一步了解元件相容性，請參閱[元件](#)。

最後，兩個 Greengrass 執行時間選項之間的選擇將取決於您的特定使用案例、裝置限制條件、功能需求和作業系統。

Greengrass 核

AWS IoT Greengrass 核是功能完整的執行時間，可讓您 AWS IoT Greengrass 在各種裝置上執行，包括閘道、伺服器 and 具有更多運算資源的邊緣裝置。如果出現下列情況，請考慮選擇 Greengrass 核：

- 運算資源：您的裝置有足夠的運算資源，例如超過 128 MB 的 RAM 和相對強大的處理器（例如，大於 1 GHz 時鐘）。
- 需要完整作業系統支援：Greengrass 核支援最廣泛的作業系統（包括大多數 Linux 反派和 Windows）。
- 元件相容性：Greengrass 核提供與服務團隊和合作夥伴發佈的 AWS IoT 現有元件最完整的相容性。

Greengrass nucleus lite

AWS IoT Greengrass nucleus lite 是一種輕量的開放原始碼執行期，可讓您 AWS IoT Greengrass 在資源受限的裝置上執行。這對於具有大量應用程式的低成本單板電腦非常有用，例如智慧家庭中樞、智慧電表、智慧車輛、邊緣 AI 和機器人。如果裝置符合下列條件，請考慮選擇 Greengrass nucleus lite：

- 資源受限：您的裝置資源有限，例如 RAM 記憶體 (512 MB 或更少)、儲存空間（快閃記憶體）或低效能處理器（少於 1 GHz）。
- 相依性限制：您的裝置廠商軟體平台不支援 Java 或 Greengrass 核所需的特定 JVM。
- 作業系統：您的裝置會執行支援 systemd 的 Linux 發行版本（例如：Ubuntu、Yocto）。

Greengrass nucleus lite 目前的限制

如 AWS IoT Greengrass v2.14.0 中所包含，Greengrass nucleus lite (v.2.0.0) 執行期提供 Greengrass nucleus (v2.14.0) 可用的功能子集。

AWS IoT Greengrass IPC（程序間通訊）機制允許元件與 Greengrass 核進行通訊。輕量版的核支援下列子集：

功能	可用性
SubscribeToTopic	Available
PublishToTopic	Available

功能	可用性
PublishToIoTCore	Available
SubscribeToIoTCore	Available
UpdateState	目前無法使用
SubscribeToComponentUpdates	目前無法使用
DeferComponentUpdate	目前無法使用
GetConfiguration	Available
UpdateConfiguration	Available
SubscribeToConfigurationUpdate	Available
SubscribeToValidateConfigurationUpdates	目前無法使用。
SendConfigurationValidityReport	目前無法使用。
GetSecretValue	目前無法使用。
PutComponentMetric	目前無法使用
GetComponentDetails	目前無法使用
RestartComponent	目前無法使用
StopComponent	目前無法使用
CreateLocalDeployment	Available
CancelLocalDeployment	目前無法使用
GetLocalDeploymentStatus	目前無法使用
ListLocalDeployments	目前無法使用
ListComponents	目前無法使用

功能	可用性
ValidateAuthorizationToken	Available
CreateDebugPassword	目前無法使用
PauseComponent	目前無法使用
ResumeComponent	目前無法使用
GetThingShadow	目前無法使用
UpdateThingShadow	目前無法使用
DeleteThingShadow	目前無法使用
ListNamedShadowsForThing	目前無法使用
SubscribeToCertificateUpdates	目前無法使用
VerifyClientDeviceIdentity	目前無法使用
GetClientDeviceAuthToken	目前無法使用
AuthorizeClientDeviceAction	目前無法使用

中的新功能 AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 是 的主要版本 AWS IoT Greengrass ，引進下列功能：

- AWS IoT Greengrass nucleus lite – AWS IoT Greengrass nucleus 2.14.0 版和更新版本提供 AWS IoT Greengrass nucleus lite。Greengrass nucleus lite 是一種輕量的開放原始碼執行期，可讓您在資源受限的裝置上執行 Greengrass。這對於具有大量應用程式的低成本單板電腦非常有用，例如智慧家庭中樞、智慧電表、智慧車輛、邊緣 AI 和機器人。如需詳細資訊，請參閱[Greengrass nucleus lite](#)。
- 發佈者支援的元件 – AWS IoT Greengrass 現在提供發佈者支援的元件。這些元件是由第三方供應商開發、提供和服務。如需詳細資訊，請參閱[發佈者支援的元件](#)。
- 在 VPC 中操作 Greengrass 裝置 – 在 VPC 中操作 Greengrass 核心裝置現已可用。這可讓您在沒有公有網際網路存取的情況下，在 VPC 中執行部署。如需詳細資訊，請參閱[在 VPC 中操作 AWS IoT Greengrass 核心裝置](#)。
- Greengrass 測試架構 (GTF) – 適用於 的 AWS IoT Greengrass Version 2 GTF 現已推出。GTF 是支援 end-to-end 自動化的建置區塊集合。它可讓 AWS IoT Greengrass Version 2 內部客戶使用與服務團隊用於合格軟體變更、自動接受和品質保證目的相同的測試架構。如需詳細資訊，請參閱 [Github 上的 Greengrass 測試架構](#)。
- PSA 認證 – 2.7.0 版及更新版本 AWS IoT Greengrass 的核心現已通過平台安全架構 (PSA) 認證。如需詳細資訊，請參閱 [AWS IoT Greengrass 已通過 PSA 認證](#)。

AWS IoT Greengrass 版本備註提供有關 AWS IoT Greengrass 版本的詳細資訊 - 新功能、更新和改進，以及一般修正。AWS IoT Greengrass 具有下列類型的版本：

- 的新功能版本 AWS IoT Greengrass
- AWS IoT Greengrass 核心軟體更新

本節包含所有 AWS IoT Greengrass V2 版本備註、最新版本，並包含主要功能變更和重大錯誤修正。如需其他次要修正的資訊，請參閱 GitHub 上的 [aws-greengrass](#) 組織。

版本備註

- [版本：2025 年 12 月 23 日 AWS IoT Greengrass 核心 v2.16.1 軟體更新](#)
- [2025 年 11 月 6 日發行：AWS IoT Greengrass Core v2.16.0 軟體更新](#)
- [2025 年 11 月 6 日發行：AWS IoT Greengrass Core v2.15.1 軟體更新](#)

- [2025 年 7 月 3 日發行：AWS IoT Greengrass Core v2.15.0 軟體更新](#)
- [2025 年 4 月 11 日發行：AWS IoT Greengrass Core v2.14.3 軟體更新](#)
- [2025 年 3 月 27 日發行：AWS IoT Greengrass Core v2.14.2 軟體更新](#)
- [版本：2025 年 2 月 7 日 AWS IoT Greengrass 核心 v2.14.1 軟體更新](#)
- [2024 年 12 月 16 日發行：AWS IoT Greengrass Core v2.14.0 軟體更新](#)
- [2024 年 8 月 26 日發行：AWS IoT Greengrass Core v2.13.0 軟體更新](#)
- [2024 年 5 月 24 日發行：AWS IoT Greengrass Core v2.12.6 軟體更新](#)
- [2024 年 4 月 25 日發行：AWS IoT Greengrass Core v2.12.5 軟體更新](#)
- [版本：2024 年 4 月 2 日 AWS IoT Greengrass Core v2.12.4 軟體更新](#)
- [2024 年 3 月 27 日發行：AWS IoT Greengrass Core v2.12.3 軟體更新](#)
- [版本：2024 年 2 月 15 日 AWS IoT Greengrass Core v2.12.2 軟體更新](#)
- [版本：2023 年 12 月 8 日 AWS IoT Greengrass 核心 v2.12.1 軟體更新](#)
- [2023 年 11 月 7 日發行：AWS IoT Greengrass Core v2.12.0 軟體更新](#)
- [2023 年 10 月 18 日發行：AWS IoT Greengrass Core v2.11.3 軟體更新](#)
- [2023 年 8 月 9 日發行：AWS IoT Greengrass Core v2.11.2 軟體更新](#)
- [2023 年 7 月 21 日發行：AWS IoT Greengrass Core v2.11.1 軟體更新](#)
- [2023 年 6 月 28 日發行：AWS IoT Greengrass Core v2.11.0 軟體更新](#)
- [2023 年 6 月 21 日發行：AWS IoT Greengrass Core v2.10.3 軟體更新](#)
- [2023 年 6 月 5 日發行：AWS IoT Greengrass Core v2.10.2 軟體更新](#)
- [2023 年 5 月 11 日發行：AWS IoT Greengrass Core v2.10.1 軟體更新](#)
- [2023 年 5 月 9 日發行：AWS IoT Greengrass Core v2.10.0 軟體更新](#)
- [2023 年 4 月 20 日發行：AWS IoT Greengrass Core v2.9.6 軟體更新](#)
- [2023 年 3 月 30 日發行：AWS IoT Greengrass Core v2.9.5 軟體更新](#)
- [版本：2023 年 2 月 24 日 AWS IoT Greengrass 核心 v2.9.4 軟體更新](#)
- [版本：2023 年 2 月 1 日 AWS IoT Greengrass 核心 v2.9.3 軟體更新](#)
- [2022 年 12 月 22 日發行：AWS IoT Greengrass Core v2.9.2 軟體更新](#)
- [2022 年 11 月 18 日發行：AWS IoT Greengrass Core v2.9.1 軟體更新](#)
- [2022 年 11 月 15 日發行：AWS IoT Greengrass Core v2.9.0 軟體更新](#)
- [2022 年 10 月 13 日發行：AWS IoT Greengrass Core v2.8.1 軟體更新](#)
- [2022 年 10 月 7 日發行：AWS IoT Greengrass Core v2.8.0 軟體更新](#)

- [2022 年 7 月 28 日發行：AWS IoT Greengrass Core v2.7.0 軟體更新](#)
- [2022 年 6 月 27 日發行：AWS IoT Greengrass Core v2.6.0 軟體更新](#)
- [2022 年 5 月 31 日發行：AWS IoT Greengrass Core v2.5.6 軟體更新](#)
- [2022 年 4 月 6 日發行：AWS IoT Greengrass Core v2.5.5 軟體更新](#)
- [版本：2022 年 3 月 23 日 AWS IoT Greengrass 核心 v2.5.4 軟體更新](#)
- [2022 年 1 月 6 日發行：AWS IoT Greengrass Core v2.5.3 軟體更新](#)
- [2021 年 12 月 3 日發行：AWS IoT Greengrass Core v2.5.2 軟體更新](#)
- [2021 年 11 月 23 日發行：AWS IoT Greengrass Core v2.5.1 軟體更新](#)
- [2021 年 11 月 12 日發行：AWS IoT Greengrass Core v2.5.0 軟體更新](#)
- [2021 年 8 月 3 日發行：AWS IoT Greengrass Core v2.4.0 軟體更新](#)
- [2021 年 6 月 29 日發行：AWS IoT Greengrass Core v2.3.0 軟體更新](#)
- [2021 年 6 月 18 日發行：AWS IoT Greengrass Core v2.2.0 軟體更新](#)
- [2021 年 4 月 26 日發行：AWS IoT Greengrass Core v2.1.0 軟體更新](#)
- [2021 年 3 月 9 日發行：AWS IoT Greengrass Core v2.0.5 軟體更新](#)
- [2021 年 2 月 4 日發行：AWS IoT Greengrass Core v2.0.4 軟體更新](#)

版本：2025 年 12 月 23 日 AWS IoT Greengrass 核心 v2.16.1 軟體更新

此版本提供 Greengrass 核元件的 2.16.1 版和 Greengrass CLI 元件的 2.16.1 版。

版本日期：2025 年 12 月 23 日

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.16.1 版已推出。 錯誤修正和改善 • 新增 Token Exchange Service 失敗後登入資料重試間隔的組態。
Greengrass CLI	Greengrass CLI 的 2.16.1 版已推出。 針對 Greengrass Nucleus 2.16.1 版更新版本。

2025 年 11 月 6 日發行：AWS IoT Greengrass Core v2.16.0 軟體更新

此版本提供 Greengrass 核元件的 2.16.0 版，並更新 AWS 提供的元件。

版本日期：2025 年 11 月 6 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.16.0 版已推出。 新功能 - 現在，Nucleus 支援 cgroups V2。 錯誤修正和改善 - 修正在取消部署時，Nucleus 不會清除過時部署的問題。
Greengrass nucleus lite	Greengrass nucleus lite 2.3.0 版已推出。 新功能 - 支援將 TPM 2.0 用於 IoT Core MQTT 授權。 - 範例 apt 套件現在支援更多作業系統：Ubuntu 22.04、Ubuntu 24.04、Debian 12 和 Debian 13。 - 現在支援 RestartComponent IPC。 錯誤修正和改善 - 本機部署不再需要網際網路存取。 - GetConfiguration 已更新，以符合 Greengrass Nucleus 執行時間行為。(重大變更) - 一般錯誤修正與改進。
Shadow 管理員	提供 2.3.12 版的影子管理員。 錯誤修正和改善 修正設定超過 1024 個裝置影子時，部署遭到封鎖的問題。
日誌管理員	日誌管理員的 2.3.11 版已推出。

元件	詳細資訊
	錯誤修正和改善 修正 Log Manager 執行期組態因上傳日誌檔案的過時資訊而無限期成長的問題。
系統日誌轉寄站	2.1.0 版的 系統日誌轉寄站 已推出。 錯誤修正和改善 - 更新元件配方以正確支援 Greengrass 核。 - 改善在沒有要上傳的日誌時的記錄輸出。 - 一般錯誤修正與改進。

2025 年 11 月 6 日發行：AWS IoT Greengrass Core v2.15.1 軟體更新

此版本提供 Greengrass 核元件的 2.15.1 版和 Greengrass CLI 元件的 2.15.1 版。

版本日期：2025 年 11 月 6 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核 2.15.1 版	錯誤修正和改善 修正在引導部署期間 30 秒關閉逾時後，Nucleus 無法清除元件程序的問題。
Greengrass CLI 2.15.1 版	針對 Greengrass Nucleus 2.15.1 版更新版本。

2025 年 7 月 3 日發行：AWS IoT Greengrass Core v2.15.0 軟體更新

此版本提供 Greengrass 核元件的 2.15.0 版，並更新 AWS 提供的元件。

版本日期：2025 年 7 月 3 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.15.0 版已推出。 新功能 - 新增遙測功能，以包含主機系統資訊，例如 CPU 和作業系統詳細資訊。如需詳細資訊，請參閱遙測指標。 - 新增deploymentConfigurationTimeSource 組態。如需詳細資訊，請參閱 Greengrass 核組態。 錯誤修正和改善 - 根據預設，Nucleus 現在會優先使用本機版本和成品，改善部署一致性並減少外部相依性。使用者可以透過在部署文件中指定替代偏好設定來覆寫此行為。 - 當提供的 Windows 使用者不符合 Windows 通常接受的字元集時，新增警告日誌。
Greengrass CLI	Greengrass CLI 的 2.15.0 版已推出。 錯誤修正和改善 - 新增修正，以便在身分驗證組態變更時，元件不會重新安裝，但仍會重新啟動。 - 一般錯誤修正與改進。
Greengrass nucleus lite	Greengrass nucleus lite 2.2.0 版已推出。 新功能 - 新增對容器映像成品 URIs 支援。 錯誤修正和改善 - 一般錯誤修正與改進。
日誌管理員	日誌管理員 的 2.3.10 版已推出。 新功能 新增組態金鑰 (updateToTlogIntervalSec)，以控制日誌上傳事件詳細資訊保留至本機交易日誌 () 的頻率config.tlog 。

元件	詳細資訊
	錯誤修正和改善 改善日誌管理員以重新整理 cloudwatch 用戶端的通訊端連線錯誤。
秘密管理員	2.2.6 版的 秘密管理員 已推出。 錯誤修正和改善 修正秘密管理員因為緩慢或沒有回應的信任平台模組而無法取得秘密的問題。

2025 年 4 月 11 日發行：AWS IoT Greengrass Core v2.14.3 軟體更新

此版本提供 Greengrass 核元件的 2.14.3 版，並更新 AWS 提供的元件。

版本日期：2025 年 4 月 11 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.14.3 版已推出。 錯誤修正和改善 • 允許權杖交換服務在連接埠組態變更時重新啟動。 • 修正機群狀態服務未觸發未鎖定 lambda 時，不會傳送元件狀態變更訊息的問題。 • 修正部署新元件版本時元件未正常關閉的問題。 • 修正內建外掛程式的注入掛鉤執行兩次導致額外的生命週期事件和重複記錄的問題。 • 改善 Windows 裝置上的元件生命週期記錄。
Greengrass CLI	Greengrass CLI 的 2.14.3 版已推出。 錯誤修正和改善 • 針對 Greengrass nucleus 2.14.3 版更新版本。
秘密管理員	2.2.4 版的秘密管理員已推出。 錯誤修正和改善 • 降低寫入本機秘密存放區的頻率。Secret Manager 現在只會在秘密更新時寫入本機存放區。
EMQX MQTT 代理程式	2.0.3 版的 EMQX MQTT 代理程式已推出。 錯誤修正和改善 • 修正如果路徑包含空格，EMQX 不會在 Windows 上啟動的問題。

2025 年 3 月 27 日發行：AWS IoT Greengrass Core v2.14.2 軟體更新

此版本提供 Greengrass 核元件的 2.14.2 版，並更新 AWS 提供的元件。

版本日期：2025 年 3 月 27 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.14.2 版已推出。 錯誤修正和改善 • 修正 HTTP 用戶端未設定交互身分驗證的問題。
Greengrass CLI	Greengrass CLI 的 2.14.2 版已推出。 錯誤修正和改善 • 針對 Greengrass nucleus 2.14.2 版更新版本。

版本：2025 年 2 月 7 日 AWS IoT Greengrass 核心 v2.14.1 軟體更新

此版本提供 Greengrass 核元件的 2.14.1 版，並更新 AWS 提供的元件。

版本日期：2025 年 2 月 7 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的 功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.14.1 版已推出。 錯誤修正和改善 • 修正 Greengrass 新安裝時元件未正確停止的問題。
Greengrass CLI	Greengrass CLI 的 2.14.1 版已推出。 錯誤修正和改善 • 針對 Greengrass nucleus 2.14.1 版更新版本。
安全通道	1.1.1 版的安全通道元件已推出。 錯誤修正和改善 • 新增支援 Greengrass nucleus lite 的組態。

2024 年 12 月 16 日發行：AWS IoT Greengrass Core v2.14.0 軟體更新

此版本提供 Greengrass 核元件的 2.14.0 版，以及新的 AWS IoT Greengrass 核精簡更新。AWS IoT Greengrass nucleus lite 是新的執行時間，適用於 AWS IoT Greengrass 第 2 版。它提供了減少記憶體使用量的替代方案。對於資源受限的裝置來說，這是很好的選項。它實作了一部分的核功能，具有針對未來版本規劃的更多特色相容性。來源碼現在可在 [Github](#) 上使用。透過 nucleus lite 執行時間，您可以：

- 將元件部署到 Greengrass 核心裝置。使用相同的配方格式，雖然某些進階功能可能尚無法使用。
- 部署為 Greengrass 元件的應用程式可以使用裝置 SDKs 來存取支援的 Greengrass IPC APIs，例如：AWS IoT Core MQTT 存取、本機 pub/sub 和 Greengrass 組態存取。如需[支援的 IPC APIs](#)清單，請參閱相容性圖表。
- 部分 AWS 受管元件已更新，支援 nucleus lite。如需現有相容[AWS 元件的清單](#)，請參閱[提供的元件](#)。

新功能：

- 使用較少的記憶體和磁碟空間（少於 5MB 的 RAM 和少於 5MB 的儲存空間）。
- 元件與主機系統的服務管理員整合（適用於目前支援的 Linux 平台）。

要注意的事項：

- AWS IoT Greengrass nucleus lite 配方區分大小寫。確保在<https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html>配方參考中使用正確的（金鑰）大小寫。
- nucleus lite 執行時間支援物件群組部署，且尚不支援（單一）Core 裝置部署目標類型。若要部署到單一 Greengrass 裝置，請使用物件群組，其中只有一個裝置。
- 核精簡型執行期使用週框記憶體資源；根據傳統執行期的使用量擴展的功能可能會因為超過精簡型資源而失敗。這包括一次最多 50 個 MQTT 訂閱的目前限制，以及配方檔案大小和部署的最大限制。如果自行編譯 lite 執行時間，則可以在編譯時設定其中一些限制。
- nucleus lite 執行時間不會隨附於 Java。若要使用需要 Java 的元件，系統將需要已安裝 Java，也可以使用元件來安裝 Java。
- 我們建議您從來源編譯 nucleus lite 執行時間，並使用專為您的系統量身打造的建置。對於 Yocto 系統，layer 可用於將 nucleus lite 執行時間整合到您的系統映像中。

- 目前，nucleus lite 假設使用 systemd 的 Linux 系統，或使用 systemd 的容器映像。
- 雖然您可以使用配方指令碼管理 Docker 容器，但 Greengrass 受管容器成品尚無法使用。
- nucleus lite 執行時間尚未支援存放在 PKCS11 模組中的金鑰。如果您的使用案例需要存放在安全元素上的金鑰，則傳統執行時間目前可支援此使用案例。為了防止裝置登入資料洩漏，請確保生產裝置使用完整磁碟加密。

除了推出 nucleus lite 之外，我們也發行了 nucleus v2.14.0。此更新為現有的 Greengrass 核帶來顯著的增強功能。

主要功能和改進：

- 新的雙堆疊端點支援可啟用 IPv6 網路通訊。
- 增強對核重新啟動失敗和目錄損毀的彈性。
- 修正 IPC PubSub 訂閱關閉中的記憶體流失問題。

版本日期：2024 年 12 月 16 日

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核精簡型	Greengrass nucleus lite 2.0.0 版已推出。

元件	詳細資訊
	新功能 - 使用較少的記憶體和磁碟空間（少於 5MB 的 RAM 和少於 5MB 的儲存空間）。 - 元件與主機系統的服務管理員整合（適用於目前支援的 Linux 平台）。
Greengrass 核	Greengrass 核的 2.14.0 版已推出。 新功能 - 新的雙堆疊端點支援可啟用 IPv6 網路通訊。 - 增強對核重新啟動失敗和目錄損毀的彈性。 錯誤修正和改善 - 修正 IPC PubSub 訂閱關閉中的記憶體流失問題。 - 修正略過條件為 true 時，因啟動逾時而進入 ERRORED 狀態之元件的執行生命週期。 - 修正當 TLS 政策設定為 TLS13_1_3_2022_10 AWS IoT Core 時，核心裝置無法連線至 的問題。
Greengrass CLI	Greengrass CLI 的 2.14.0 版已推出。 錯誤修正和改善 在 cli 命令中驗證部署目標參數。
串流管理員	串流管理員的 2.14.0 版已推出。 新功能 - 新增啟動逾時的新組態金鑰。預設值為 120 秒。 - 新增 Greengrass nucleus lite 的配方支援。
MQTT 5 代理程式 (EMQX)	2.0.2 版的 MQTT 5 代理程式 (EMQX) 已推出。 錯誤修正和改善 修正 EMQX 在用戶端裝置驗證元件就緒之前啟動的問題。

元件	詳細資訊
Lambda 執行時間元件	Lambda 執行期元件的 2.0.9 版已推出。 錯誤修正和改善 • 使用 Python 3.12 修正語法警告
Lambda 管理員元件	Lambda 管理員元件的 2.3.5 版已推出。 錯誤修正和改善 • 可使用 epoll 而非 nio 來改善效能
Secret Manager 元件	2.2.2 版的 Secret Manager 元件已推出。 錯誤修正和改善 • 修正秘密管理員未下載設定部分 arn 之秘密的問題。
安全通道元件	1.1.0 版的安全通道元件已推出。 新功能 • 新增 Greengrass nucleus lite 的配方支援
CloudWatch 指標元件	CloudWatch 指標元件的 1.1.0 版已推出。 新功能 • 新增 Greengrass nucleus lite 的配方支援

2024 年 8 月 26 日發行：AWS IoT Greengrass Core v2.13.0 軟體更新

此版本提供 Greengrass 核元件的 2.13.0 版。

版本日期：2024 年 8 月 26 日

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.13.0 版已推出。 新功能 - 支援《Nucleus》中的 FIPS 端點。如需詳細資訊，請參閱FIPS 端點。 錯誤修正和改善 - 取消部署改進 - 部署現在可以在合併新組態時以及等待服務啟動時取消。
串流管理員	串流管理員元件的 2.1.13 版已推出。 錯誤修正和改善 在 中支援 FIPS 端點 AWS IoT SiteWise
秘密管理員	Secret Manager 元件的 2.2.0 版已推出。 新功能 - 新增透過新元件組態金鑰定期重新整理已設定秘密的支援。 - 在 GetSecretValue IPC 請求中新增對新請求參數的支援，以為每個請求重新整理秘密
IP 偵測器	IP 偵測器元件的 2.2.0 版已推出。 新功能 新增 IPv6 的支援。您現在可以將 IPv6 用於本機傳訊。

元件	詳細資訊
用戶端裝置身分驗證	用戶端裝置驗證的 2.5.1 版已推出。 錯誤修正和改善 • 一般錯誤和修正。 • 支援 FIPS 端點。
本機偵錯主控台	2.4.3 版的 Local debug 主控台 已推出。 錯誤修正和改善 • 修正錯誤地以 Mbps 而非每秒位元組顯示 STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH 的問題。

2024 年 5 月 24 日發行：AWS IoT Greengrass Core v2.12.6 軟體更新

此版本提供 Greengrass 核元件的 2.12.6 版，並更新 AWS 提供的元件。

版本日期：2024 年 5 月 24 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.12.6 版已推出。 錯誤修正和改善 修正在特定 ARMv8 處理器上造成啟動時當機的問題，包括 Jetson Nano。
Greengrass CLI	Greengrass CLI 的 2.12.6 版已推出。 錯誤修正和改善 針對 Greengrass nucleus 2.12.6 版更新版本。
秘密管理員	2.1.8 版的秘密管理員已推出。 錯誤修正和改善 修正秘密管理員不接受部分 arn 的問題。

2024 年 4 月 25 日發行：AWS IoT Greengrass Core v2.12.5 軟體更新

此版本提供 Greengrass 核元件的 2.12.5 版，並更新 AWS 提供的元件。

版本日期：2024 年 4 月 25 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.5 版已推出。 錯誤修正和改善 - 修正部署轉返偶爾在轉返具有硬相依性的先前損壞元件時卡住的問題。 - 修正機群佈建後，核不會發佈狀態更新的問題。 - 收到 404 錯誤後，新增 GetDeploymentConfiguration API 的重試次數。

版本：2024 年 4 月 2 日 AWS IoT Greengrass Core v2.12.4 軟體更新

此版本提供 Greengrass 核元件的 2.12.4 版，並更新 AWS 提供的元件。

版本日期：2024 年 4 月 2 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.12.4 版已推出。 錯誤修正和改善 修正在某些 Linux 裝置上啟動時，核心進入死鎖條件的問題。

2024 年 3 月 27 日發行：AWS IoT Greengrass Core v2.12.3 軟體更新

此版本提供 Greengrass 核元件的 2.12.3 版，並更新 AWS 提供的元件。

版本日期：2024 年 3 月 27 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修

補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.3 版已推出。 錯誤修正和改善 - 修正核心在重新啟動後和元件復原期間未報告正確元件狀態的問題。 - 一般錯誤修正與改進。
Shadow 管理員	影子管理員元件的 2.3.7 版已推出。 錯誤修正和改善 修正陰影管理員在陰影管理員同步期間定期記錄NullPointerException 錯誤的問題。
機群佈建	1.2.1 版的AWS IoT 機群佈建外掛程式已推出。 錯誤修正和改善 修正 Greengrass 核啟動期間機群佈建外掛程式離線的問題。機群佈建外掛程式現在會無限期重試 MQTT 連線呼叫。
IP 偵測器	磁碟多工緩衝處理器元件的 2.1.9 版已推出。 錯誤修正和改善 調整 IP 取得的步驟，以僅在偵錯日誌層級傳送日誌。
Moquette MQTT 3.1.1 代理程式元件	Moquette MQTT 3.1.1 代理程式元件的 2.3.6 版已推出。 錯誤修正和改善 一般錯誤修正與改進。

元件	詳細資訊
Lambda 管理員	Lambda 管理員元件 的 2.3.3 版已推出。 錯誤修正和改善 一般錯誤修正與改進。
本機偵錯主控台	本機偵錯主控台元件 的 2.4.2 版已推出。 錯誤修正和改善 一般錯誤修正與改進。

版本：2024 年 2 月 15 日 AWS IoT Greengrass Core v2.12.2 軟體更新

此版本提供 Greengrass 核元件的 2.12.2 版，並更新 AWS 提供的元件。

版本日期：2024 年 2 月 15 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.12.2 版已推出。 錯誤修正和改善 - 修正舊日誌未正確清除的問題。 - 一般錯誤修正與改進。
Shadow 管理員	2.3.6 版的影子管理員元件已推出。 錯誤修正和改善 修正重新取得連線後，裝置離線時透過 AWS 雲端 更新刪除的影子屬性繼續存在於本機影子中的問題。
Lambda 啟動器	2.0.13 版的 lambda 啟動器元件已推出。 錯誤修正和改善 一般錯誤修正與改進。
磁碟多工緩衝處理	磁碟多工緩衝處理器元件的 1.0.3 版已推出。 錯誤修正和改善 透過重複使用資料庫連線來改善效能。

版本：2023 年 12 月 8 日 AWS IoT Greengrass 核心 v2.12.1 軟體更新

此版本提供 Greengrass 核元件的 2.12.1 版，並更新 AWS 提供的元件。

版本日期：2023 年 12 月 8 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.12.1 版已推出。 錯誤修正和改善 修正核心可能會將 MQTT 訂閱複製到部署主題而導致額外記錄和 MQTT 發佈的問題。
用戶端裝置身分驗證	用戶端裝置驗證元件 的 2.4.5 版已推出。 新功能 新增對萬用字元字首的支援，以使用 <code>selectionRule</code> 參數選取物件名稱。 錯誤修正和改善 修正在特定情況下，憑證未以新的連線資訊更新的問題。
磁碟多工緩衝處理	磁碟多工緩衝處理器元件 的 1.0.2 版已推出。 錯誤修正和改善 修正在某些情況下未保留 MQTT 訊息格式欄位的問題。

元件	詳細資訊
MQTT 橋接器	磁碟多工緩衝處理器元件的 2.3.1 版 已推出。 錯誤修正和改善 修正本機 MQTT 用戶端進入中斷連線迴圈的問題。
串流管理員	串流管理員元件 的 2.1.12 版已推出。 錯誤修正和改善 更新使用登入資料的順序，以便 Greengrass 登入資料優先於 AWS 服務請求。

2023 年 11 月 7 日發行：AWS IoT Greengrass Core v2.12.0 軟體更新

此版本提供 Greengrass 核元件的 2.12.0 AWS 版，以及所提供元件的更新。

版本日期：2023 年 11 月 7 日

版本重點

- 在復原時引導 – AWS IoT Greengrass 現在提供名為 `Greengrass 核組態參數BootstrapOnRollback`。此功能可讓您在復原部署中執行引導生命週期步驟。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修

補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.12.0 版已推出。
	新功能
	可讓您在復原部署中執行引導生命週期步驟。

2023 年 10 月 18 日發行：AWS IoT Greengrass Core v2.11.3 軟體更新

此版本提供 Greengrass 核元件的 2.11.3 版。

發佈日期：2023 年 10 月 18 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.11.3 版已推出。 錯誤修正和改善 - 修正核心中可能在元件相依性失敗時不當啟動元件的問題。 新功能 - 新增可設定的 s3 端點類型。
Lambda 管理員	Lambda 管理員元件的 2.3.1 版已推出。 錯誤修正和改善 調整特定錯誤的日誌層級。
Local Debug 主控台	2.4.0 版的 Lambda 管理員元件已推出。 新功能 新增串流管理員偵錯主控台。
日誌管理員	日誌管理員元件的 2.3.6 版已推出。 錯誤修正和改善 調整特定錯誤的日誌層級。
Shadow 管理員	Shadow Manager 元件的 2.3.4 版已推出。 錯誤修正和改善 新增對 null 和空白陰影狀態文件的支援。

2023 年 8 月 9 日發行：AWS IoT Greengrass Core v2.11.2 軟體更新

此版本提供 Greengrass 核元件的 2.11.2 版。

版本日期：2023 年 8 月 9 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.11.2 版已推出。 錯誤修正和改善 • 修正使用大量 (> 50) 訂閱時，可能離線的 nucleus MQTT 5 用戶端問題。 • 新增對 Docker 撥號 TCP 失敗的重試。

2023 年 7 月 21 日發行：AWS IoT Greengrass Core v2.11.1 軟體更新

此版本提供 Greengrass 核元件的 2.11.1 版。

版本日期：2023 年 7 月 21 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.11.1 版已推出。 錯誤修正和改善 • 修正當引導任務失敗且部署中繼資料檔案損毀時，核心未啟動的問題。 • 修正部署狀態更新中未報告隨需 Lambda 元件的問題。 • 新增對重複授權政策 IDs 支援。
Lambda 管理員	Lambda 管理員的 2.2.11 版已推出。

元件	詳細資訊
	錯誤修正和改善 修正 Lambda 組態變更時 LegacySubscriptionRouter 組態未更新的問題。

2023 年 6 月 28 日發行：AWS IoT Greengrass Core v2.11.0 軟體更新

此版本提供 Greengrass 核元件的 2.11.0 版。

版本日期：2023 年 6 月 28 日

版本重點

- 持久性磁碟多工緩衝處理 – AWS IoT Greengrass 現在為從 Greengrass 核心裝置多工緩衝處理的訊息提供持久性多工緩衝處理實作 AWS IoT Core。此元件會將這些傳出訊息存放在磁碟上。如需詳細資訊，請參閱[磁碟多工緩衝處理](#)。
- 本機部署改善 – 您現在可以取消本機部署、設定部署失敗處理政策，以及取得詳細的部署狀態。
- 日誌速度改善 – 日誌管理員元件的日誌上傳速度已改善。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.11.0 版已推出。 新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的故障處理政策。 • 新增對磁碟多工緩衝處理外掛程式的支援。
Greengrass CLI	Greengrass CLI 的 2.11.0 版已推出。 新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的故障處理政策。 • 改善詳細的部署狀態報告。
磁碟多工緩衝處理	1.0.0 版的磁碟多工緩衝處理器元件已推出。 • 磁碟多工緩衝處理器元件提供從 Greengrass 核心裝置傳送至 之訊息的持久性儲存 AWS IoT Core。
日誌管理員	日誌管理員元件的 2.3.5 版已推出。 改進 改善日誌上傳速度。

2023 年 6 月 21 日發行：AWS IoT Greengrass Core v2.10.3 軟體更新

此版本提供 Greengrass 核元件的 2.10.3 版。

版本日期：2023 年 6 月 21 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的 功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.10.3 版已推出。 錯誤修正和改善 • 修正 Greengrass 在使用 PKCS#11 供應商時未訂閱部署通知的問題。

2023 年 6 月 5 日發行：AWS IoT Greengrass Core v2.10.2 軟體更新

此版本提供 Greengrass 核元件的 2.10.2 版。

版本日期：2023 年 6 月 5 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.10.2 版已推出。 錯誤修正和改善 • 允許不區分大小寫的元件生命週期剖析。 • 修正環境 PATH 變數未正確重新建立的問題。 • 修正元件的代理 URI 編碼，包括具有特殊字元之使用者名稱的串流管理員。
用戶端裝置身分驗證	用戶端裝置驗證元件的 2.4.2 版已推出。 新功能 新增startupTimeoutSeconds 組態選項。
Lambda 管理員	Lambda 管理員元件的 2.2.9 版已推出。 錯誤修正和改善 修正連接埠號碼因時鐘偏斜而損毀的問題。
日誌管理員	日誌管理員元件的 2.3.4 版已推出。

元件	詳細資訊
	錯誤修正和改善 - 新增將 <code>periodicUploadIntervalSec</code> 參數設定為分數的支援。最小值為 1 微秒。 - 修正日誌管理員不遵守 CloudWatch <code>putLogEvents</code> 限制的問題。
MQTT 3.1 代理程式 (Moquette)	2.3.3 版的 MQTT 3.1 代理程式 (Moquette) 元件已推出。 新功能 新增 <code>startupTimeoutSeconds</code> 組態選項。
MQTT 橋接器	2.2.6 版的 MQTT 橋接器 元件已推出。 新功能 新增 <code>startupTimeoutSeconds</code> 組態選項。
串流管理員	串流管理員 元件的 2.1.7 版已推出。 錯誤修正和改善 修正串流管理員無法正確讀取代理組態的問題。

2023 年 5 月 11 日發行：AWS IoT Greengrass Core v2.10.1 軟體更新

此版本提供 Greengrass 核元件的 2.10.1 版。

版本日期：2023 年 5 月 11 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的 功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.10.1 版已推出。 錯誤修正和改善 - 修正在特定 ARMv8 處理器上啟動時可能導致當機的問題，包括 Jetson Nano。 - Greengrass 不再關閉元件的標準，這會將行為還原為 pre-2.10.0 行為
串流管理員	2.1.6 版的新串流管理員已推出。 錯誤修正和改善 修正在特定 ARMv8 處理器上啟動時可能導致當機的問題，包括 Jetson Nano。

2023 年 5 月 9 日發行：AWS IoT Greengrass Core v2.10.0 軟體更新

此版本提供 Greengrass 核元件的 2.10.0 版，並更新 AWS 提供的元件。

版本日期：2023 年 5 月 9 日

版本重點

- MQTT5 支援 – AWS IoT Greengrass 現在支援 AWS IoT Core 使用 MQTT5 從 傳送和接收訊息。如需詳細資訊，請參閱[發佈 AWS IoT Core MQTT 訊息](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS ，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.10.0 版已推出。 新功能 • 新增對空白規則表達式的interpolateComponentConfiguration 支援。Greengrass 現在會從根組態物件插入 。 • 新增對 MQTT5 的支援。 • 新增無需掃描即可快速載入外掛程式元件的機制。 • 讓 Greengrass 透過刪除未使用的 Docker 映像來節省磁碟空間。 錯誤修正和改善 • 修正轉返從部署保留特定組態值的問題。

元件	詳細資訊
	- 修正 Greengrass 核在自訂非AWS 憑證和資料端點中驗證 AWS 網域序列的問題。 - 更新多群組相依性解析，以透過 AWS 雲端 交涉重新解析所有群組相依性，而不是鎖定作用中的版本。此更新也會移除部署錯誤碼 <code>INSTALLED_COMPONENT_NOT_FOUND</code>。 - 更新 Greengrass 核，以便在 Docker 映像已存在於本機時略過下載。 - 在逾時到期之前，更新 Greengrass 核以重新啟動元件安裝步驟。 - 其他次要修正和改進。
Shadow 管理員	2.3.2 版的新影子管理員已推出。 錯誤修正和改善 修正當本機影子資料庫損毀時，影子管理員進入 BROKEN 狀態的問題。

2023 年 4 月 20 日發行：AWS IoT Greengrass Core v2.9.6 軟體更新

此版本提供 Greengrass 核元件的 2.9.6 版。

發佈日期：2023 年 4 月 20 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.6 版已推出。 錯誤修正和改善 修正 Greengrass 部署失敗並顯示 LAUNCH_DIRECTORY_C ORRUPTED 錯誤，且後續裝置重新啟動無法啟動 Greengrass 的問題。當您在部署需要 Greengrass 重新啟動的多個物件群組之間移動 Greengrass 裝置時，可能會發生此錯誤。

2023 年 3 月 30 日發行：AWS IoT Greengrass Core v2.9.5 軟體更新

此版本提供 Greengrass 核元件的 2.9.5 版。

版本日期：2023 年 3 月 30 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.5 版已推出。 新功能 • 新增對 Greengrass 核軟體簽章驗證的支援。 錯誤修正和改善 • 修正當本機配方中繼資料區域與 Greengrass 核啟動區域不相符時，部署失敗的問題。發生這種情況時，Greengrass 核現在會與雲端重新交涉。 • 修正 MQTT 訊息多工緩衝處理器填滿且永遠不會移除訊息的問題。 • 其他次要修正和改進。

版本：2023 年 2 月 24 日 AWS IoT Greengrass 核心 v2.9.4 軟體更新

此版本提供 Greengrass 核元件的 2.9.4 版。

版本日期：2023 年 2 月 24 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修

補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.4 版已推出。 錯誤修正和改善 • 在捨棄 QOS 0 訊息之前檢查 null 訊息。 • 如果任務狀態詳細資訊值超過 1024 個字元限制，則會將其截斷。 • 如果該路徑包含空格，請更新 Windows 的引導指令碼，以正確讀取 Greengrass 根路徑。 • 更新訂閱，AWS IoT Core 以便在未傳送訂閱回應時捨棄用戶端訊息。 • 確保當主要組態檔案損毀或遺失時，核會從備份檔案載入其組態。

版本：2023 年 2 月 1 日 AWS IoT Greengrass 核心 v2.9.3 軟體更新

此版本提供 Greengrass 核元件的 2.9.3 版。

版本日期：2023 年 2 月 1 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 提供的元件 AWS，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.3 版已推出。 錯誤修正和改善 • 確保 MQTT IDs 不會重複。 • 新增更強大的檔案讀取和寫入，以避免並從損毀中復原。 • 重試 Docker 映像提取特定網路相關錯誤。 • 新增 MQTT 連線noProxyAddresses 的選項。

2022 年 12 月 22 日發行：AWS IoT Greengrass Core v2.9.2 軟體更新

此版本提供 Greengrass 核元件的 2.9.2 版。

版本日期：2022 年 12 月 22 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.9.2 版已推出。 錯誤修正和改善 - 修正設定 <code>interpolateComponentConfiguration</code> 不適用於進行中部署的問題。 - 使用 OSHI 列出所有子程序。

2022 年 11 月 18 日發行：AWS IoT Greengrass Core v2.9.1 軟體更新

此版本提供 Greengrass 核元件的 2.9.1 版，並更新 AWS 提供的元件。

版本日期：2022 年 11 月 18 日

版本重點

- 日誌管理員 – 日誌管理員現在會處理並直接上傳作用中日誌檔案，而不是等待輪換新檔案。這項改善可大幅減少日誌延遲。如需詳細資訊，請參閱[日誌管理員](#)

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.9.1 版已推出。 錯誤修正和改善 當部署移除外掛程式元件時，新增 Greengrass 重新啟動的修正。
日誌管理員	2.3.0 版的新日誌管理員已推出。  Note 當您升級至 log Manager 2.3.0 時，我們建議您升級至 Greengrass 核 2.9.1。 新功能 - 透過處理和直接上傳作用中日誌檔案，而不是等待輪換新檔案，以減少日誌延遲。 錯誤修正和改善 - 改善使用唯一名稱輪換檔案時的日誌輪換支援。 - 其他次要修正和改進。

2022 年 11 月 15 日發行：AWS IoT Greengrass Core v2.9.0 軟體更新

此版本提供 Greengrass 核心元件的 2.9.0 版，並更新 AWS 提供的元件。

版本日期：2022 年 11 月 15 日

版本重點

- 離線身分驗證 – AWS IoT Greengrass 現在支援離線身分驗證。您可以設定 AWS IoT Greengrass 核心裝置，讓用戶端裝置可以連線至核心裝置，即使核心裝置未連線至雲端也一樣。如需詳細資訊，請參閱[離線身分驗證](#)。
- 子部署 – 您現在可以建立子部署。您可以使用子部署來解決失敗的部署。每個子部署都可以在較小的裝置子集上測試失敗部署的不同組態。如需詳細資訊，請參閱[建立子部署](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核心更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.9.0 版已推出。

元件	詳細資訊
	新功能 - 新增建立子部署的功能，以使用較小的裝置子集重試部署。此功能會建立更有效率的方式來測試和解決失敗的部署。 錯誤修正和改善 - 改善對沒有 useradd、groupadd 和 之系統的支援 usermod。 - 其他次要修正和改進。
用戶端裝置身分驗證	用戶端裝置驗證元件 的 2.3.0 版已推出。 新功能 - 新增用戶端裝置的離線身分驗證支援。透過此功能，當核心裝置未連線至網際網路時，用戶端裝置可以繼續連線至核心裝置。 - 新增對客戶提供的憑證授權單位 (CA) 的支援。您的核心裝置使用客戶提供的 CA 做為根憑證，以產生 MQTT 代理程式憑證。
MQTT 5 代理程式 (EMQX)	MQTT 5 代理程式 (EMQX) 元件的 1.2.0 版已推出。 新功能 新增憑證鏈的支援。
Moquette MQTT 代理程式	2.3.0 版的新 Moquette MQTT 代理程式元件 已推出。 新功能 新增憑證鏈的支援。
秘密管理員	2.1.4 版的新 秘密管理員 已推出。 錯誤修正和改善 修正部署秘密管理員並重新啟動 Greengrass 核時，快取秘密遭到移除的問題。

元件	詳細資訊
串流管理員	2.1.2 版的新 串流管理員 已推出。 錯誤修正和改善 修正使用非英文語言的 Windows 作業系統問題。

2022 年 10 月 13 日發行：AWS IoT Greengrass Core v2.8.1 軟體更新

此版本提供 Greengrass 核元件的 2.8.1 版。

版本日期：2022 年 10 月 13 日

Note

如果您使用的是 Greengrass 核版本 2.8.0，強烈建議您升級至 Greengrass 核版本 2.8.1。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.8.1 版已推出。 錯誤修正和改善 • 修正 Greengrass API 錯誤未正確產生部署錯誤代碼的問題。 • 修正當元件在部署期間達到 ERRORED 狀態時，機群狀態更新會傳送不正確資訊的問題。 • 修正 Greengrass 有超過 50 個現有訂閱時，部署無法完成的問題。

2022 年 10 月 7 日發行：AWS IoT Greengrass Core v2.8.0 軟體更新

此版本提供 Greengrass 核元件的 2.8.0 版，以及 MQTT 5 代理程式 (EMQX) 元件的 1.1.0 版。

版本日期：2022 年 10 月 7 日

版本重點

- 部署錯誤代碼 – Greengrass 核現在會報告[部署運作狀態](#)回應，其中包含元件部署無法完成時的詳細錯誤代碼。如需詳細資訊，請參閱[詳細部署錯誤代碼](#)。
- 元件錯誤狀態 – Greengrass 核現在會報告[元件運作狀態](#)回應，其中包含元件進入 BROKEN 或 狀態時的詳細錯誤 ERRORED 狀態。如需詳細資訊，請參閱[詳細的元件狀態碼](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修

補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.8.0 版已推出。 新功能 - 更新 Greengrass 核以報告部署運作狀態回應，其中包含在將元件部署至核心裝置時的詳細錯誤代碼。如需詳細資訊，請參閱詳細部署錯誤代碼。 - 更新 Greengrass 核以報告元件運作狀態回應，其中包含元件進入 BROKEN 或 ERRORED 狀態時的詳細錯誤代碼。如需詳細資訊，請參閱詳細的元件狀態碼。 - 展開狀態訊息欄位，以改善裝置的雲端可用性資訊。 - 改善機群狀態服務穩健性。 錯誤修正和改善 - 允許損壞的元件在其組態變更時重新安裝。 - 修正在引導部署期間重新啟動核心導致部署失敗的問題。 - 修正根路徑包含空格時安裝失敗的 Windows 問題。 - 修正元件在部署期間關閉使用新版本的關閉指令碼的問題。 - 各種關機改進。 - 其他次要修正和改進。
MQTT 5 代理程式 (EMQX)	MQTT 5 代理程式 (EMQX) 元件的 1.1.0 版已推出。 新功能 - 新增對 EMQX 組態的支援，包括代理程式選項和外掛程式。 錯誤修正和改善 - 將 EMQX 更新至 4.4.9 版。

2022 年 7 月 28 日發行：AWS IoT Greengrass Core v2.7.0 軟體更新

此版本提供 Greengrass 核元件的 2.7.0 版、串流管理員元件的 2.1.0 版，以及 Lambda 管理員元件的 2.2.5 版。

發行日期：2022 年 7 月 28 日

版本重點

- 串流管理員遙測指標 – 串流管理員現在會自動將遙測指標傳送至 Amazon EventBridge，因此您可以建立雲端應用程式來監控和分析核心裝置上傳的資料量。如需詳細資訊，請參閱[從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)。
- 自訂憑證授權機構 (CA) – AWS IoT 現在支援由自訂憑證 CA 簽署的用戶端憑證，其中 CA 未註冊。如需詳細資訊，請參閱[使用私有 CA 簽署的裝置憑證](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.7.0 版已推出。

元件	詳細資訊
	新功能 - 當核心裝置套用本機部署時，更新 Greengrass 核以傳送狀態更新至 AWS IoT Greengrass 雲端。 - 新增支援由自訂憑證授權單位 (CA) 簽署的用戶端憑證，其中 CA 未註冊 AWS IoT。若要使用此功能，您可以將新的greengrassDataPlaneEndpoint 組態選項設定為 iotdata。如需詳細資訊，請參閱使用私有 CA 簽署的裝置憑證。 錯誤修正和改善 - 修正在特定情況下，Greengrass 核停止或重新啟動時復原部署的問題。核現在會在核重新啟動後繼續部署。 - 當您指定 將軟體設定為系統服務時，請更新 Greengrass 安裝程式以遵守 --start 引數。 - 更新的行為SubscribeToComponentUpdates，以在核心更新元件的事件中設定部署 ID。 - 其他次要修正和改進。
串流管理員	串流管理員元件的 2.1.0 版已推出。 新功能 - 更新此元件以自動將遙測指標傳送至 Amazon EventBridge。如需詳細資訊，請參閱從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料。 此功能需要 2.7.0 版或更新版本的 Greengrass 核元件。 - 針對 Greengrass nucleus 2.7.0 版更新版本。
Lambda 管理員	Lambda 管理員元件的 2.2.5 版已推出。 新功能 - 在您訂閱本機發佈/訂閱訊息的事件來源中新增對 MQTT 主題萬用字元的支援。 此功能需要 2.6.0 版或更新版本的 Greengrass 核元件。 - 針對 Greengrass nucleus 2.7.0 版更新版本。

2022 年 6 月 27 日發行：AWS IoT Greengrass Core v2.6.0 軟體更新

此版本提供 Greengrass 核心元件的 2.6.0 版、新 AWS 提供的元件，以及 AWS 所提供元件的更新。

版本日期：2022 年 6 月 27 日

版本重點

- 本機發佈/訂閱主題中的萬用字元 – 您現在可以在訂閱本機發佈/訂閱主題時使用 MQTT 萬用字元。如需詳細資訊，請參閱[發佈/訂閱本機訊息](#)及[SubscribeToTopic](#)。
- 用戶端裝置影子支援 – 您現在可以在自訂元件中與用戶端裝置影子互動，並使用 同步用戶端裝置影子 AWS IoT Core。如需詳細資訊，請參閱[與用戶端裝置影子互動和同步](#)。
- 本機 MQTT 5 支援用戶端裝置 – 您現在可以部署 EMQX MQTT 5 代理程式，以在用戶端裝置和核心裝置之間使用 MQTT 5 功能進行通訊。如需詳細資訊，請參閱[MQTT 5 代理程式 \(EMQX\)](#)及[將用戶端裝置連接至核心裝置](#)。
- 元件組態中的配方變數 – 您現在可以在元件組態中使用特定的配方變數。當您在配方中定義元件的預設組態，或在部署中設定元件時，您可以使用這些配方變數。如需詳細資訊，請參閱[配方變數](#)及在[合併更新中使用配方變數](#)。
- IPC 授權政策中的萬用字元 – 您現在可以使用 * 萬用字元來比對程序間通訊 (IPC) 授權政策中的任意字元組合。此萬用字元可讓您在單一授權政策中允許存取多個資源。如需詳細資訊，請參閱[授權政策中的萬用字元](#)。
- 管理本機部署和元件的 IPC 操作 – 您現在可以開發自訂元件來管理本機部署並檢視元件詳細資訊。如需詳細資訊，請參閱 [IPC：管理本機部署和元件](#)。
- 驗證和授權用戶端裝置的 IPC 操作 – 您現在可以使用這些操作來建立自訂本機代理程式元件。如需詳細資訊，請參閱 [IPC：驗證和授權用戶端裝置](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.6.0 版已推出。 新功能 - 當您訂閱本機發佈/訂閱主題時，新增對 MQTT 萬用字元的支援。如需詳細資訊，請參閱發佈/訂閱本機訊息及SubscribeToTopic。 - 新增對元件組態中配方變數的支援，配方<code>component_dependency_name</code>:configuration: <code>json_pointer</code> 變數除外。當您在配方DefaultConfiguration 中定義元件的 或在部署中設定元件時，您可以使用這些配方變數。若要啟用此功能，請將 interpolateComponentConfiguration 組態選項設定為 true。如需詳細資訊，請參閱配方變數及在合併更新中使用配方變數。 - 在程序間通訊 (IPC) 授權政策中新增*萬用字元的完整支援。您現在可以在資源字串中指定*字元，以符合字元的任意組合。如需詳細資訊，請參閱授權政策中的萬用字元。 - 新增對自訂元件的支援，以呼叫 Greengrass CLI 使用的 IPC 操作。您可以使用這些 IPC 操作來管理本機部署、檢視元件詳細資訊，以及產生可用來登入本機偵錯主控台的密碼。如需詳細資訊，請參閱IPC：管理本機部署和元件。 錯誤修正和改善 - 修正相依元件在特定情況下重新啟動或變更狀態時無法反應的問題。 - 改善核心裝置在部署失敗時向 AWS IoT Greengrass 雲端服務報告的錯誤訊息。

元件	詳細資訊
	- 修正在特定情況下，Greengrass 核重新啟動時套用物件部署兩次的問題。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。
MQTT 5 代理程式 (EMQX)	1.0.0 版的新 EMQX MQTT 5 代理程式元件 已推出。 新功能 新增對本機 EMQX MQTT 5 代理程式的支援。用戶端裝置可以連線至此 MQTT 代理程式，以使用 MQTT 5 功能與核心裝置通訊。
Shadow 管理員	提供影子管理員元件的 2.2.0 版。 新功能 - 透過本機發佈/訂閱界面新增對本機影子服務的支援。您現在可以在影子 MQTT 主題上與本機發佈/訂閱訊息代理程式通訊，以取得、更新和刪除核心裝置上的影子。此功能可讓您使用 MQTT 橋接器，在用戶端裝置與本機發佈/訂閱界面之間轉送影子主題的訊息，將用戶端裝置連線至本機影子服務。 此功能需要 2.6.0 版或更新版本的 Greengrass 核元件。若要將用戶端裝置連線至本機影子服務，您還必須使用 MQTT 橋接器元件 的 v2.2.0 或更新版本。 - 新增您可以設定的 direction 選項，以自訂方向，在本機影子服務與之間同步影子 AWS 雲端。您可以設定此選項，以減少頻寬和對的連線 AWS 雲端。
用戶端裝置身分驗證	用戶端裝置驗證元件 的 2.2.0 版已推出。 新功能 - 新增自訂元件的支援，以呼叫程序間通訊 (IPC) 操作來驗證和授權用戶端裝置。例如，您可以在自訂 MQTT 代理程式元件中使用這些操作。如需詳細資訊，請參閱 IPC：驗證和授權用戶端裝置。 - 新增您可以設定來調整此元件執行方式的 maxActiveAuthToken s cloudQueueSize 、 和 threadPoolSize 選項。

元件	詳細資訊
MQTT 橋接器	2.2.0 版的 MQTT 橋接器元件 已推出。 新功能 - 當您將本機發佈/訂閱指定為來源訊息代理程式時，新增對 MQTT 主題萬用字元 (# 和 +) 的支援。 此功能需要 2.6.0 版或更新版本的 Greengrass 核元件。 - 新增 選項，您可以指定此 targetTopicPrefix 選項來設定 MQTT 橋接器，以在轉送訊息時將字首新增至目標主題。
Greengrass CLI	Greengrass CLI 的 2.6.0 版已推出。 新功能 - 新增對自訂元件的支援，以呼叫 Greengrass CLI 使用的處理程序間通訊 (IPC) 操作。您可以使用這些 IPC 操作來管理本機部署、檢視元件詳細資訊，以及產生可用來登入 本機偵錯主控台 的密碼。如需詳細資訊，請參閱 IPC：管理本機部署和元件。 錯誤修正和改善 - 其他次要修正和改進。

2022 年 5 月 31 日發行：AWS IoT Greengrass Core v2.5.6 軟體更新

此版本提供 Greengrass 核元件的 2.5.6 版和日誌管理員元件的 2.2.4 版。

版本日期：2022 年 5 月 31 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.6 版已推出。 新功能 - 新增對使用 ECC 金鑰之硬體安全模組的支援。您可以使用硬體安全模組 (HSM) 安全地存放裝置的私有金鑰和憑證。如需詳細資訊，請參閱硬體安全整合。 錯誤修正和改善 - 修正在特定情況下以中斷的安裝指令碼部署元件時，部署從未完成的問題。 - 改善啟動期間的效能。 - 其他次要修正和改進。
日誌管理員	日誌管理員元件的 2.2.4 版已推出。 錯誤修正和改善 - 改善處理無效組態時的穩定性。 - 其他次要修正和改進。

2022 年 4 月 6 日發行：AWS IoT Greengrass Core v2.5.5 軟體更新

此版本提供 Greengrass 核元件的 2.5.5 版。

版本日期：2022 年 4 月 6 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.5 版已推出。 新功能 • 新增元件GG_ROOT_CA_PATH 的環境變數，讓您可以存取自訂元件中的根憑證授權機構 (CA) 憑證。 錯誤修正和改善 • 新增對使用英文以外顯示語言之 Windows 裝置的支援。 • 更新 Greengrass 核如何剖析布林值安裝程式引數，因此您可以指定不含布林值的布林值引數來指定true值。例如，您現在可以指定 --provision true，--provision 而不是使用自動資源佈建安裝。 • 修正核心裝置在特定情況下佈建後未向 AWS IoT Greengrass 雲端服務報告其狀態的問題。 • 其他次要修正和改進。

版本：2022 年 3 月 23 日 AWS IoT Greengrass 核心 v2.5.4 軟體更新

此版本提供 Greengrass 核元件的 2.5.4 版，以及 Lambda 啟動器元件的 2.0.10 版。

版本日期：2022 年 3 月 23 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.5.4 版已推出。 錯誤修正和改善 • 一般錯誤修正與改進。
Lambda 啟動器	Lambda 啟動器 元件的 2.0.10 版已推出。 錯誤修正和改善 • 一般錯誤修正與改進。

2022 年 1 月 6 日發行：AWS IoT Greengrass Core v2.5.3 軟體更新

此版本提供 2.5.3 版的 Greengrass 核元件和新的 PKCS#11 提供者元件。

版本日期：2022 年 1 月 6 日

版本重點

- 硬體安全整合 - 您現在可以將 AWS IoT Greengrass Core 軟體設定為使用您安全地存放在硬體安全模組 (HSM) 中的私有金鑰和憑證。如需詳細資訊，請參閱[硬體安全整合](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.5.3 版已推出。
	新功能
	新增硬體安全整合的支援。您可以使用硬體安全模組 (HSM) 安全地存放裝置的私有金鑰和憑證。如需詳細資訊，請參閱硬體安全整合。

元件	詳細資訊
	錯誤修正和改善 修正 核心與 建立 MQTT 連線時，執行時間例外狀況的問題 AWS IoT Core。
PKCS#11 供應商	PKCS#11 供應商元件的 2.0.0 版已推出。 新功能 新增硬體安全整合的支援。您可以使用硬體安全模組 (HSM) 安全地存放裝置的私有金鑰和憑證。如需詳細資訊，請參閱硬體安全整合。

2021 年 12 月 3 日發行：AWS IoT Greengrass Core v2.5.2 軟體更新

此版本提供 Greengrass 核元件的 2.5.2 版。

版本日期：2021 年 12 月 3 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.2 版已推出。 錯誤修正和改善 修正 Greengrass 核更新後，Windows 服務在您停止或重新啟動裝置後無法再次啟動的問題。
AWS IoT Device Defender	AWS IoT Device Defender 元件的 3.0.1 版已推出。 此版本的 AWS IoT Device Defender 元件預期組態參數與 2.x 版不同。如果您使用 2.x 版的非預設組態，而且想要從 v2.x 升級到 v3.x，則必須更新元件的組態。如需詳細資訊，請參閱AWS IoT Device Defender 元件組態。 新功能 - 新增對執行 Windows 之核心裝置的支援。 - 將元件類型從 Lambda 元件變更為一般元件。此元件現在不再依賴舊版訂閱路由器元件來建立訂閱。 - 新增新的UseInstaller 組態參數，可讓您選擇性地停用安裝元件相依性的安裝指令碼。

2021 年 11 月 23 日發行：AWS IoT Greengrass Core v2.5.1 軟體更新

此版本提供 Greengrass 核元件的 2.5.1 版。

版本日期：2021 年 11 月 23 日

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.1 版已推出。 錯誤修正和改善 • 新增對 Windows 上 32 位元版本的 Java 執行期環境 (JRE) 的支援。 • 變更政策 AWS IoT 未授予greengrass:ListThingGroupsForCoreDevice 許可之核心裝置的物件群組移除行為。在此版本中，部署會繼續、記錄警告，而且當您從物件群組移除核心裝置時，不會移除元件。如需詳細資訊，請參閱將 AWS IoT Greengrass 元件部署至裝置。 • 修正 Greengrass 核提供給 Greengrass 元件程序的系統環境變數問題。您現在可以重新啟動元件以使用最新的系統環境變數。

2021 年 11 月 12 日發行：AWS IoT Greengrass Core v2.5.0 軟體更新

此版本提供 Greengrass 核元件的 2.5.0 版、新 AWS 提供的元件，以及 AWS 所提供元件的更新。

版本日期：2021 年 11 月 12 日

版本重點

- Windows 裝置支援 - 您現在可以在執行 Windows 作業系統的裝置上執行 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱[Greengrass 功能相容性](#)。

- 新物件群組移除行為 — 您現在可以從物件群組移除核心裝置，以在下次部署至該裝置時移除該物件群組的元件。

Important

由於此變更，核心裝置的 AWS IoT 政策必須具有 `greengrass:ListThingGroupsForCoreDevice` 許可。如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，預設 AWS IoT 政策會允許 `greengrass:*`，其中包含此許可。如需詳細資訊，請參閱[裝置身分驗證和授權 AWS IoT Greengrass](#)。

- 硬體安全支援 - 您現在可以將 AWS IoT Greengrass Core 軟體設定為使用硬體安全模組 (HSM)，以便安全地存放裝置的私有金鑰和憑證。如需詳細資訊，請參閱[硬體安全整合](#)。
- HTTPS 代理支援 - 您現在可以將 AWS IoT Greengrass Core 軟體設定為透過 HTTPS 代理連線。如需詳細資訊，請參閱[連線至連接埠 443 或透過網路代理](#)。

版本詳細資訊

- [平台支援更新](#)
- [公有元件更新](#)

平台支援更新

平台	詳細資訊
Windows	AWS IoT Greengrass 現在支援在下列版本的 Windows 上執行 AWS IoT Greengrass 核心軟體： • Windows 10 • Windows Server 2019 如需詳細資訊，請參閱Greengrass 功能相容性。

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.5.0 版已推出。 新功能 • 新增對執行 Windows 之核心裝置的支援。 • 變更物件群組移除的行為。使用此版本，您可以從物件群組中移除核心裝置，以在下一次部署中解除安裝該物件群組的元件。 由於此變更，核心裝置的 AWS IoT 政策必須具有 <code>greengrass::ListThingGroupsForCoreDevice</code> 許可。如果您使用 AWS IoT Greengrass Core 軟體安裝程式佈建資源，預設 AWS IoT 政策會允許 <code>greengrass:*</code>，其中包含此許可。如需詳細資訊，請參閱裝置身分驗證和授權 AWS IoT Greengrass。 • 新增對 HTTPS 代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理。 • 新增新的 <code>windowsUser</code> 組態參數。您可以使用此參數來指定預設使用者，以用來在 Windows 核心裝置上執行元件。如需詳細資訊，請參閱設定執行元件的使用者。 • 新增可用於自訂 HTTP 請求逾時的新 <code>httpClient</code> 組態選項，以改善慢速網路的效能。如需詳細資訊，請參閱 httpClient 組態參數。 錯誤修正和改善 • 修正引導生命週期選項，以從元件重新啟動核心裝置。 • 在配方變數中新增連字號的支援。

元件	詳細資訊
	• 修正隨需 Lambda 函數元件的 IPC 授權。 • 改善日誌訊息，並將非關鍵日誌從 INFO 變更為DEBUG關卡，因此日誌更實用。 • 從 Greengrass 核在安裝具有自動佈建的 AWS IoT Greengrass Core 軟體時建立的預設權杖交換角色中移除iot:DescribeCertificate 許可。Greengrass 核不會使用此許可。 • 修正問題，以便在 iam:CreatePolicy 適用於相同政策時，自動佈建指令碼不需要 iam:GetPolicy 許可。 • 其他次要修正和改進。
Greengrass CLI	Greengrass CLI 的 2.5.0 版已推出。 新功能 • 新增對執行 Windows 之核心裝置的支援。 • 新增您可以指定的新AuthorizedWindowsGroups 組態參數，以授權系統群組在 Windows 裝置上使用 Greengrass CLI。 • 新增本機部署的 windowsUser 參數。您可以使用此參數指定要用來在 Windows 核心裝置上執行元件的使用者。

元件	詳細資訊
CloudWatch 指標	CloudWatch 指標元件的 3.0.0 版已推出。 此版本的 CloudWatch 指標元件預期與 2.x 版不同的組態參數。如果您使用 2.x 版的非預設組態，而且想要從 v2.x 升級到 v3.x，則必須更新元件的組態。如需詳細資訊，請參閱 CloudWatch 指標元件組態。 新功能 • 新增對執行 Windows 之核心裝置的支援。 • 將元件類型從 Lambda 元件變更為一般元件。此元件現在不再依賴舊版訂閱路由器元件來建立訂閱。 • 新增InputTopic 組態參數，以指定元件訂閱接收訊息的主題。 • 新增OutputTopic 組態參數，以指定元件發佈狀態回應的主題。 • 新增PubSubToIoTCore 組態參數，以指定是否發佈和訂閱 AWS IoT Core MQTT 主題。 • 新增新的UseInstaller 組態參數，可讓您選擇性地停用安裝元件相依性的安裝指令碼。 錯誤修正和改善 新增對輸入資料中重複時間戳記的支援。
Lambda 管理員	Lambda 管理員元件的 2.2.0 版已推出。 錯誤修正和改善 • 修正 Lambda 函數在重新啟動後無法寫入日誌的問題。 • 修正當主題中有萬用字元時，舊版訂閱路由器傳送重複訊息的問題。 • 修正非固定 Lambda 函數無法在 中使用 Greengrass 處理程序間通訊 (IPC) 程式庫的問題 AWS IoT Device SDK。

2021 年 8 月 3 日發行：AWS IoT Greengrass Core v2.4.0 軟體更新

此版本提供 Greengrass 核元件的 2.4.0 版、新 AWS 提供的元件，以及 AWS 所提供元件的更新。

版本日期：2021 年 8 月 3 日

版本重點

- 系統資源限制 - Greengrass 核心元件現在支援系統資源限制。您可以設定每個元件程序可在核心裝置上使用的 CPU 和 RAM 用量上限。如需詳細資訊，請參閱[設定元件的系統資源限制](#)。
- 暫停/繼續元件 - Greengrass 核心現在支援暫停和繼續元件。您可以使用程序間通訊 (IPC) 程式庫來開發自訂元件，以暫停和繼續其他元件的程序。如需詳細資訊，請參閱[PauseComponent](#)及[ResumeComponent](#)。
- 使用 AWS IoT 機群佈建安裝 - 使用新的 AWS IoT 機群佈建外掛程式，在連線至以 AWS IoT 佈建必要 AWS 資源的裝置上安裝 AWS IoT Greengrass 核心軟體。裝置使用宣告憑證來佈建。您可以在製造期間將宣告憑證內嵌在裝置上，因此每個裝置都可以在上線後立即佈建。如需詳細資訊，請參閱[使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體](#)。
- 使用自訂佈建安裝 - 開發自訂佈建外掛程式，以便在裝置上安裝 AWS IoT Greengrass 核心軟體時佈建所需的 AWS 資源。您可以建立在安裝期間執行的 Java 應用程式，為您的自訂使用案例設定 Greengrass 核心裝置。如需詳細資訊，請參閱[使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核心更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.4.0 版已推出。 新功能 • 新增對系統資源限制的支援。您可以設定每個元件程序可在核心裝置上使用的 CPU 和 RAM 用量上限。如需詳細資訊，請參閱設定元件的系統資源限制。 • 新增 IPC 操作以暫停和繼續元件。如需詳細資訊，請參閱PauseComponent及ResumeComponent。 • 新增佈建外掛程式的支援。您可以指定要在安裝期間執行的 JAR 檔案，以佈建 Greengrass 核心裝置所需的 AWS 資源。Greengrass 核包含一個介面，您可以實作此介面來開發自訂佈建外掛程式。如需詳細資訊，請參閱使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。 • 將選用thing-name-policy 引數新增至 AWS IoT Greengrass Core 軟體安裝程式。當您使用自動資源佈建安裝 AWS IoT Greengrass 核心軟體時，可以使用此選項指定現有或自訂 AWS IoT 政策。 錯誤修正和改善 • 在啟動時更新記錄組態。這修正了啟動時未套用記錄組態的問題。 • 更新 nucleus 載入器符號連結，以在安裝期間指向 Greengrass 根資料夾中的元件存放區。此更新可讓您刪除安裝 AWS IoT Greengrass 核心軟體時下載的 JAR 檔案和其他核子成品。 • 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。
Greengrass CLI	Greengrass CLI 的 2.4.0 版已推出。 新功能 • 新增對系統資源限制的支援。建立本機部署時，您可以設定每個元件程序可在核心裝置上使用的 CPU 和 RAM 用量上限。如需詳細資訊，請參閱設定元件的系統資源限制和 部署建立命令。
AWS IoT 依宣告佈建機群	宣告外掛程式的 AWS IoT 機群佈建現已可用。如需詳細資訊，請參閱使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體。

元件	詳細資訊
	新功能 • 新增使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體的支援。在安裝期間，裝置會連線至 AWS IoT 以佈建所需的 AWS 資源，並下載裝置憑證以用於一般操作。

2021 年 6 月 29 日發行：AWS IoT Greengrass Core v2.3.0 軟體更新

此版本提供 Greengrass 核元件的 2.3.0 版。

版本日期：2021 年 6 月 29 日

版本重點

- 大型組態支援 - Greengrass 核元件現在支援高達 10 MB 的部署文件。您現在可以將較大的組態更新部署到 Greengrass 元件。

Note

若要使用此功能，核心裝置的 AWS IoT 政策必須允許 `greengrass:GetDeploymentConfiguration` 許可。如果您使用 [AWS IoT Greengrass 核心軟體安裝程式佈建資源](#)，則核心裝置的 AWS IoT 政策會允許 `greengrass:*`，其中包含此許可。如需詳細資訊，請參閱[裝置身分驗證和授權 AWS IoT Greengrass](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.3.0 版已推出。 新功能 - 新增對部署組態文件的支援，最高可達 10 MB，從 7 KB（針對目標物件的部署）或 31 KB（針對目標物件群組的部署）開始。 若要使用此功能，核心裝置的 AWS IoT 政策必須允許 <code>greengrass:GetDeploymentConfiguration</code> 許可。如果您使用 AWS IoT Greengrass 核心軟體安裝程式佈建資源，則核心裝置的 AWS IoT 政策會允許 <code>greengrass:*</code>，其中包含此許可。如需詳細資訊，請參閱裝置身分驗證和授權 AWS IoT Greengrass。 - 新增 <code>iot:thingName</code> 配方變數。您可以使用此配方變數來取得配方中核心裝置 AWS IoT 物件的名稱。如需詳細資訊，請參閱配方變數。 錯誤修正和改善 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。

2021 年 6 月 18 日發行：AWS IoT Greengrass Core v2.2.0 軟體更新

此版本提供 Greengrass 核元件的 2.2.0 版、新 AWS 提供的元件，以及 AWS 所提供元件的更新。

版本日期：2021 年 6 月 18 日

版本重點

- 用戶端裝置支援 AWS- 新提供的用戶端裝置元件可讓您使用雲端探索將用戶端裝置連接至核心裝置。您可以在 Greengrass 元件中同步用戶端裝置 AWS IoT Core，並與用戶端裝置互動。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。
- 本機影子服務 - 新的影子管理員元件會在您的核心裝置上啟用本機影子服務。您可以在離線時使用中的 Greengrass 處理程序間通訊 (IPC) 程式庫，使用此影子服務與本機影子互動 AWS IoT Device SDK。您也可以使用影子管理員元件來同步本機影子狀態 AWS IoT Core。如需詳細資訊，請參閱[與裝置陰影互動](#)。

版本詳細資訊

- [公有元件更新](#)

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.2.0 版已推出。 新功能 - 新增本機陰影管理的 IPC 操作。 錯誤修正和改善 - 減少 JAR 檔案的大小。

元件	詳細資訊
	- 減少記憶體用量。 - 修正在某些情況下未更新日誌組態的問題。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。
Shadow 管理員	2.0.0 版的新影子管理員元件已推出。 新功能 - 新增對傳統和具名陰影的支援。 - 新增使用 IPC 進行本機陰影管理的支援。 - 新增與進行陰影同步的支援 AWS IoT Core。
用戶端裝置身分驗證	2.0.0 版的新用戶端裝置驗證元件已推出。 新功能 - 新增對 Greengrass 用戶端裝置的支援，這是透過 MQTT 連線至核心裝置的本機 IoT 裝置。 - 新增用戶端裝置及其 MQTT 動作的身分驗證和授權支援。
Moquette MQTT 代理程式	2.0.0 版的新Moquette MQTT 代理程式元件已推出。 新功能 新增對處理與用戶端裝置通訊之本機 Moquette MQTT 代理程式的支援。
MQTT 橋接器	2.0.0 版的新MQTT 橋接器元件已推出。 新功能 新增支援以在本機 MQTT 代理程式、本機 Greengrass 發佈/訂閱代理程式和 AWS IoT Core MQTT 代理程式之間轉送訊息。
IP 偵測器	2.0.0 版的新IP 偵測器元件已推出。 新功能 新增支援，將核心裝置的本機 MQTT 代理程式端點報告至 AWS IoT Greengrass 雲端服務，供用戶端裝置連線。

元件	詳細資訊
日誌管理員	日誌管理員元件的 2.1.1 版已推出。 錯誤修正和改善 修正系統日誌組態在特定情況下未更新的問題。
DLR 物件偵測	2.1.2 版的 DLR 物件偵測已推出。 錯誤修正和改善 修正導致範例 DLR 物件偵測推論結果中有不正確週框方塊的影像擴展問題。
TensorFlow Lite 物件偵測	TensorFlow Lite 物件偵測的 2.1.1 版已推出。 錯誤修正和改善 修正導致範例 TensorFlow Lite 物件偵測推論結果中有不正確週框方塊的影像擴展問題。

2021 年 4 月 26 日發行：AWS IoT Greengrass Core v2.1.0 軟體更新

此版本提供 Greengrass 核心元件的 2.1.0 版，並更新 AWS 提供的元件。

版本日期：2021 年 4 月 26 日

版本重點

- Docker Hub 和 Amazon Elastic Container Registry (Amazon ECR) 整合 - 新的 Docker 應用程式管理員元件可讓您從 Amazon ECR 下載公有或私有映像。您也可以使用此元件從 Docker Hub 和 下載公有映像 AWS Marketplace。如需詳細資訊，請參閱[執行 Docker 容器](#)。
- 適用於 AWS IoT Greengrass Core 軟體的 Dockerfile 和 Docker 映像 - 您可以使用 Greengrass Docker 映像在使用 AWS IoT Greengrass Amazon Linux 2 作為基礎作業系統的 Docker 容器中執行。您也可以使用 AWS IoT Greengrass Dockerfile 來建置自己的 Greengrass 映像。如需詳細資訊，請參閱[在 Docker 容器中執行 AWS IoT Greengrass 核心軟體](#)。

- 支援其他機器學習架構和平台 - 您可以部署使用預先訓練模型的範例機器學習推論元件，以使用 TensorFlow Lite 2.5.0 和 DLR 1.6.0 執行範例影像分類和物件偵測。此版本也擴展了對 Armv8 (AArch64) 裝置的範例機器學習支援。如需詳細資訊，請參閱[執行機器學習推論](#)。

版本詳細資訊

- [平台支援更新](#)
- [公有元件更新](#)

平台支援更新

平台	詳細資訊
Docker	的 Dockerfile 和 Docker 映像 AWS IoT Greengrass 現已可用。 Dockerfile AWS IoT Greengrass 提供 Dockerfile 來建置容器映像，其具有安裝在 Amazon Linux 2 (x86_64) 基礎映像上的 AWS IoT Greengrass 核心軟體和相依性。您可以修改 Dockerfile 中的基礎映像，以在不同的平台架構 AWS IoT Greengrass 上執行。 Docker 映像檔 AWS IoT Greengrass 提供預先建置的 Docker 映像，其具有安裝在 Amazon Linux 2 (x86_64) 基礎映像上的 AWS IoT Greengrass 核心軟體和相依性。 如需詳細資訊，請參閱在 Docker 容器中執行 AWS IoT Greengrass 核心軟體。

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核的 2.1.0 版已推出。 新功能 - 支援從 Amazon ECR 中的私有儲存庫下載 Docker 映像。 - 新增下列參數來自訂核心裝置上的 MQTT 組態： <code>maxInFlightPublishes</code> – 可同時傳輸的未確認 MQTT QoS 1 訊息數量上限。 <code>maxPublishRetry</code> – 重試無法發佈之訊息的次數上限。 - 新增<code>fleetstatusservice</code> 組態參數，以設定核心裝置將裝置狀態發佈至 的間隔 AWS 雲端。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。 錯誤修正和改善 - 修正核重新啟動時導致陰影部署重複的問題。 - 修正當發生服務載入例外狀況時，導致核心當機的問題。 - 改善元件相依性解析，使包含循環相依性的部署失敗。 - 修正如果先前已從核心裝置移除該元件，導致外掛程式元件無法重新部署的問題。 - 修正導致HOME環境變數設定為 <code>/greengrass/v2 /work</code> Lambda 元件目錄或執行為根目錄的元件的問題。HOME 變數現在已正確設定為執行元件之使用者的主目錄。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。

元件	詳細資訊
Docker 應用程式管理員	2.0.0 版的新 Docker 應用程式管理員元件 已推出。 新功能 • 管理登入資料，以從 Amazon ECR 中的私有儲存庫下載映像。 • 從 Amazon ECR、Docker Hub 和 下載公有映像 AWS Marketplace。
Lambda 啟動器	Lambda 啟動器元件 的 2.0.4 版已推出。 錯誤修正和改善 • 修正元件未正確傳遞 AddGroupOwner 至 Lambda 函數容器的問題。
傳統訂閱路由器	已提供 舊版訂閱路由器元件 的 2.1.0 版。 錯誤修正和改善 • 新增支援以指定元件名稱，而不是 source 和 ARNstarget。如果您指定訂閱的元件名稱，則不需要在每次 Lambda 函數版本變更時重新設定訂閱。
本機偵錯主控台	2.1.0 版的 本機偵錯主控台元件 已推出。 新功能 • 使用 HTTPS 來保護與本機偵錯主控台的連線。HTTPS 預設為啟用。 錯誤修正和改善 • 您可以在組態編輯器中關閉閃爍列訊息。

元件	詳細資訊
日誌管理員	日誌管理員元件的 2.1.0 版已推出。 錯誤修正和改善 • 使用 <code>logFileDirectoryPath</code> 和 的預設值 <code>logFileRegex</code>，適用於列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件。 • 將日誌上傳至 CloudWatch Logs 時，透過設定的網路代理正確路由流量。 • 正確處理日誌串流名稱中的冒號字元 (:)。CloudWatch Logs 日誌串流名稱不支援冒號。 • 從日誌串流中移除物件群組名稱，以簡化日誌串流名稱。 • 移除在正常行為期間列印的錯誤日誌訊息。
DLR 影像分類	DLR 映像分類元件的 2.1.1 版已推出。 新功能 • 使用深度學習執行期 v1.6.0。 • 新增對 Armv8 (AArch64) 平台上範例映像分類的支援。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。 • 為範例推論啟用攝影機整合。使用新的 <code>UseCamera</code> 組態參數，讓範例推論程式碼存取 Greengrass 核心裝置上的攝影機，並在擷取的影像上本機執行推論。 • 新增將推論結果發佈至 的支援 AWS 雲端。使用新的 <code>PublishResultsOnTopic</code> 組態參數來指定您要發佈結果的主題。 • 新增新的 <code>ImageDirectory</code> 組態參數，可讓您為要在其中執行推論的映像指定自訂目錄。 錯誤修正和改善 • 將推論結果寫入元件日誌檔案，而非單獨的推論檔案。 • 使用 AWS IoT Greengrass 核心軟體記錄模組來記錄元件輸出。 • 使用 AWS IoT Device SDK 讀取元件組態並套用組態變更。

元件	詳細資訊
DLR 物件偵測	DLR 物件偵測 元件的 2.1.1 版已推出。 新功能 • 使用深度學習執行期 v1.6.0。 • 新增對 Armv8 (AArch64) 平台上範例物件偵測的支援。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。 • 為範例推論啟用攝影機整合。使用新的 UseCamera 組態參數，讓範例推論程式碼存取 Greengrass 核心裝置上的攝影機，並在擷取的影像上本機執行推論。 • 新增將推論結果發佈至 的支援 AWS 雲端。使用新的 PublishResultsOnTopic 組態參數來指定您要發佈結果的主題。 • 新增新的 ImageDirectory 組態參數，可讓您為要在其中執行推論的映像指定自訂目錄。 錯誤修正和改善 • 將推論結果寫入元件日誌檔案，而非單獨的推論檔案。 • 使用 AWS IoT Greengrass 核心軟體記錄模組來記錄元件輸出。 • 使用 AWS IoT Device SDK 讀取元件組態並套用組態變更。
DLR 影像分類模型存放區	DLR 影像分類模型存放 區元件的 2.1.1 版已推出。 新功能 • 新增 Armv8 (AArch64) 平台的範例 ResNet-50 影像分類模型。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。
DLR 物件偵測模型存放區	DLR 物件偵測模型存放 區元件的 2.1.1 版已推出。 新功能 • 新增 Armv8 (AArch64) 平台的範例 YOLOv3 物件偵測模型。AArch64 這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。

元件	詳細資訊
DLR 安裝程式	DLR ???元件的 1.6.1 版已推出。 新功能 - 安裝 Deep Learning Runtime v1.6.0 及其相依性。 - 新增在 Armv8 (AArch64) 平台上安裝 DLR 的支援。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。 錯誤修正和改善 - AWS IoT Device SDK 在虛擬環境中安裝，以讀取元件組態並套用組態變更。 - 其他次要錯誤修正和改善。
TensorFlow Lite 影像分類	新 TensorFlow Lite 影像分類元件的 2.1.0 版已推出。 新功能 使用 TensorFlow Lite 新增對範例影像分類推論的支援。
TensorFlow Lite 物件偵測	新 TensorFlow Lite 物件偵測元件的 2.1.0 版已推出。 新功能 使用 TensorFlow Lite 新增範例物件偵測推論的支援。
TensorFlow Lite 影像分類模型存放區	2.1.0 版的新 TensorFlow Lite 影像分類模型存放區元件已推出。 新功能 使用 TensorFlow Lite 為範例影像分類推論提供預先訓練的 MobileNet v1 量化模型。
TensorFlow Lite 物件偵測模型存放區	新 TensorFlow Lite 物件偵測模型存放區元件的 2.1.0 版已推出。 新功能 提供在 COCO 資料集上訓練的預先訓練單一鏡頭偵測 (SSD) MobileNet 模型，以使用 TensorFlow Lite 進行範例物件偵測推論。

元件	詳細資訊
TensorFlow Lite	新 TensorFlow Lite 元件的 2.5.0 版已推出。 新功能 在 Armv7, Armv8 (AArch64) 和 x86_64 平台上的虛擬環境中安裝 TensorFlow Lite 1.6.0 版及其相依性。

2021 年 3 月 9 日發行：AWS IoT Greengrass Core v2.0.5 軟體更新

此版本提供 Greengrass 核心元件的 2.0.5 AWS 版，以及更新提供的元件。它修正了網路代理支援的問題，以及 AWS 中國區域中 Greengrass 資料平面端點的問題。

版本日期：2021 年 3 月 9 日

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核心更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.0.5 版已推出。

元件	詳細資訊
	錯誤修正和改善 • 下載 AWS 提供的元件時，透過設定的網路代理正確路由流量。 • 在中國 AWS 區域使用正確的 Greengrass 資料平面端點。

2021 年 2 月 4 日發行：AWS IoT Greengrass Core v2.0.4 軟體更新

此版本提供 Greengrass 核心元件的 2.0.4 版。它包含新 `greengrassDataPlanePort` 參數，以透過連接埠 443 設定 HTTPS 通訊並修正錯誤。使用執行 AWS IoT Greengrass Core 軟體安裝程式 `sts:GetCallerIdentity` 時，最小 IAM 政策現在需要 `iam:GetPolicy` 和 `--provision true`。

版本日期：2021 年 2 月 4 日

公有元件更新

下表列出 AWS 提供的元件，其中包含新的和更新的功能。

Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	詳細資訊
Greengrass 核	Greengrass 核 的 2.0.4 版已推出。

元件	詳細資訊
	新功能 • 透過連接埠 443 啟用 HTTPS 流量。您可以使用 2.0.4 版 nucleus 元件的新 <code>greengrassDataPlanePort</code> 組態參數，將 HTTPS 通訊設定為透過連接埠 443 傳輸，而非預設連接埠 8443。如需詳細資訊，請參閱透過連接埠 443 設定 HTTPS。 • 新增工作路徑配方變數。您可以使用此配方變數來取得元件工作資料夾的路徑，可用來在元件及其相依性之間共用檔案。如需詳細資訊，請參閱工作路徑配方變數。 錯誤修正和改善 • 如果角色政策已存在，則防止建立字符交換 AWS Identity and Access Management (IAM) 角色政策。 由於此變更，安裝程式現在在使用執行 <code>sts:GetCallerIdentity</code> 時需要 <code>iam:GetPolicy</code> 和 <code>--provision true</code>。如需詳細資訊，請參閱安裝程式佈建資源的最低 IAM 政策。 • 正確處理尚未成功註冊的部署取消。 • 更新組態，以在復原部署時移除具有較新時間戳記的較舊項目。 • 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。

從第 1 AWS IoT Greengrass 版遷移

AWS IoT Greengrass Version 2 是 AWS IoT Greengrass 核心軟體、APIs 和 主控台的主要版本。AWS IoT Greengrass V2 引進了數個改善 AWS IoT Greengrass V1 項目，例如模組化應用程式、部署到大型裝置機群，以及支援其他平台。

Note

終止支援通知：2026 年 10 月 7 日 AWS 將終止 的支援 AWS IoT Greengrass Version 1。2026 年 10 月 7 日之後，您將無法再存取 AWS IoT Greengrass V1 主控台或 AWS IoT Greengrass V1 資源。

請遵循本指南中的指示從 遷移 AWS IoT Greengrass V1 至 AWS IoT Greengrass V2。

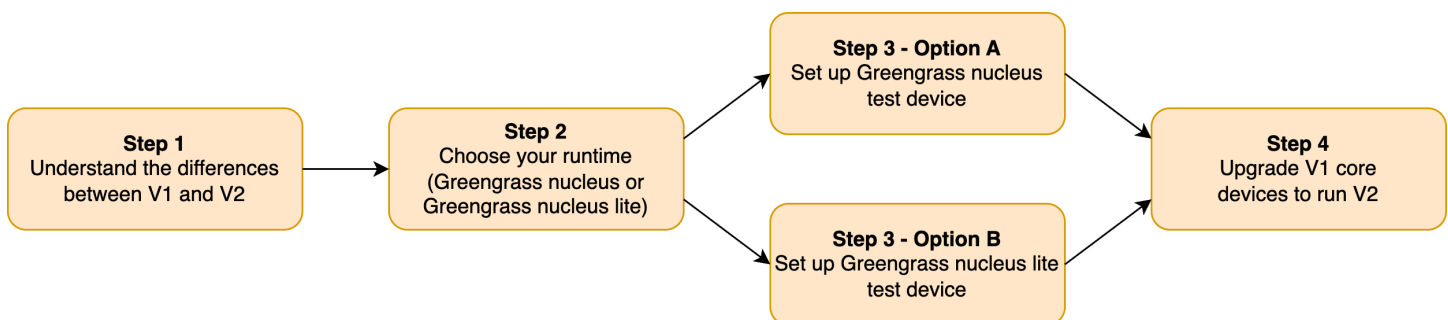
遷移概觀

在高階，您可以使用下列程序將核心裝置從 升級至 AWS IoT Greengrass V1 AWS IoT Greengrass V2。

在遷移之前，您將在兩個執行時間選項之間進行選擇：

- Greengrass 核（ 遷移工作量較低、功能完整支援 ）
- Greengrass nucleus lite（ 更高的遷移工作量，專為資源受限的裝置而設計 ）。

您遵循的確切程序取決於您的裝置資源、必要功能和特定環境需求。



1. [了解 V1 和 V2 之間的差異](#)

AWS IoT Greengrass V2 引進裝置機群和可部署軟體的新基本概念，而 V2 可簡化 V1 的數個概念。

AWS IoT Greengrass V2 雲端服務和 AWS IoT Greengrass Core 軟體 v2.x 無法與 AWS IoT Greengrass V1 雲端服務和 AWS IoT Greengrass Core 軟體 v1.x 回溯相容。因此，AWS IoT Greengrass V1 over-the-air(OTA) 更新無法將核心裝置從 V1 升級到 V2。

2. [選擇您的執行時間 \(Greengrass nucleus 或 Greengrass nucleus lite\)](#)

根據裝置資源和功能需求，在 Greengrass 核或 Greengrass 核精簡之間決定：

- Greengrass 核路徑：降低遷移工作量。Lambda 函數可以匯入為 Lambda 元件，且程式碼變更最少。支援 V1 功能（本機陰影服務、用戶端裝置、連接器）。
- Greengrass 核精簡路徑：更高的遷移工作量。Lambda 函數需要轉換為一般元件，需要程式碼變更才能使用 AWS IoT Device SDK V2/AWS IoT Greengrass 元件 SDK，而不是 AWS IoT Greengrass Core SDK。不支援本機影子服務、用戶端裝置或連接器。

3. [設定新裝置以測試 V2 上的 V1 應用程式 V2](#)

若要將生產環境中裝置的風險降至最低，請建立新的裝置，以在 V2 上測試 V1 應用程式。V2 根據您的執行時間選擇，選擇設定指南：

- 選項 A - Greengrass 核執行時間：[設定新裝置以在 V2 上測試 V1 應用程式 V2](#)。以最小程式碼變更的 Lambda 元件匯入 Lambda 函數。
- 選項 B - Greengrass nucleus lite 執行時間：[設定新裝置以在 V2 \(Greengrass nucleus lite\) 上測試 V1 應用程式 V2](#)。使用將 Lambda 函數轉換為一般元件 AWS IoT Device SDK。

4. [升級 V1 核心裝置以執行 V2](#)

在新裝置上測試之後，請升級現有的 V1 核心裝置，以執行 AWS IoT Greengrass Core 軟體 v2.x 和 AWS IoT Greengrass V2 元件。若要將裝置機群從 V1 遷移至 V2，請針對機群中的每個裝置重複此步驟。

AWS IoT Greengrass V1 與之間的差異 AWS IoT Greengrass V2

AWS IoT Greengrass V2 為裝置、機群和可部署的軟體引進新的基本概念。本節說明 V1 中不同的 V2 概念。

Greengrass 概念和術語

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
應用程式碼	在 中 AWS IoT Greengrass V1，Lambda 函數定義在核心裝置上執行的軟體。在每個 Greengrass 群組中，您會定義函數使用的訂閱和本機資源。對於 AWS IoT Greengrass Core 軟體在容器化 Lambda 執行時間環境中執行的 Lambda 函數，您可以定義容器參數，例如記憶體限制。	在 中 AWS IoT Greengrass V2，元件是在核心裝置上執行的軟體模組。 • 每個元件都有配方，定義元件的中繼資料、參數、相依性和指令碼，以在元件生命週期中的每個步驟執行。 • 配方也會定義元件的成品，也就是二進位檔案，例如指令碼、編譯程式碼和靜態資源。 • 當您將元件部署到核心裝置時，核心裝置會下載元件配方和成品來執行元件。 您可以將 V1 Lambda 函數匯入為在 Lambda 執行期環境中執行的元件 AWS IoT Greengrass V2。當您匯入 Lambda 函數時，您可以指定函數的訂閱、本機資源和容器參數。如需詳細資訊，請參閱步驟 2：建立和部署 AWS IoT Greengrass V2 元件以遷移 AWS IoT Greengrass V1 應用程式。 如需如何建立自訂元件的詳細資訊，請參閱開發 AWS IoT Greengrass 元件。
AWS IoT Greengrass 群組和部署	在 中 AWS IoT Greengrass V1，群組會定義核心裝置、該	在 中 AWS IoT Greengrass V2，您可以使用部署來定義在

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
	核心裝置的設定和軟體，以及可連線至該核心裝置的 AWS IoT 物件清單。您可以建立部署，將群組的組態傳送至核心裝置。	核心裝置上執行的軟體元件和組態。 • 每個部署都以單一核心裝置（即 AWS IoT 物件）或可包含多個核心裝置的 AWS IoT 物件群組為目標。 • 部署至物件群組是連續的，因此當您將核心裝置新增至物件群組時，它會接收該群組的軟體組態。 如需詳細資訊，請參閱將 AWS IoT Greengrass 元件部署至裝置。 在 中 AWS IoT Greengrass V2，您也可以使用 Greengrass CLI 來建立本機部署，以在您的開發它們的裝置上測試自訂軟體元件。如需詳細資訊，請參閱建立 AWS IoT Greengrass 元件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 核心軟體	在 中 AWS IoT Greengrass V1，AWS IoT Greengrass Core 軟體是包含軟體及其所有功能的單一套件。您安裝 AWS IoT Greengrass Core 軟體的邊緣裝置稱為 Greengrass 核心。	在 中 AWS IoT Greengrass V2，AWS IoT Greengrass 核心軟體是模組化的，因此您可以選擇要安裝的項目來控制記憶體佔用空間。 • Greengrass 核元件是 AWS IoT Greengrass Core 軟體的最低必要安裝。您安裝核的邊緣裝置稱為 Greengrass 核心裝置。 • 核會處理核心裝置上其他元件的部署、協調和生命週期管理。 • 串流管理員、秘密管理員和日誌管理員等功能，都是只有在您需要這些功能時才能部署的元件。如需詳細資訊，請參閱AWS提供的元件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
連接器	在 中 AWS IoT Greengrass V1，連接器是預先建置的模組，您可以部署到 AWS IoT Greengrass V1 核心裝置，以與本機基礎設施 AWS、裝置通訊協定和其他雲端服務互動。	在 中 AWS IoT Greengrass V2，AWS 提供實作 V1 中連接器所提供功能的 Greengrass 元件。下列 AWS IoT Greengrass V2 元件提供 Greengrass V1 連接器功能： • CloudWatch 指標元件 • AWS IoT Device Defender 元件 • Firehose 元件 • Modbus-RTU 通訊協定轉接器元件 • Amazon SNS 元件 如需詳細資訊，請參閱AWS 提供的元件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
連線裝置 (Greengrass 裝置)	在 中 AWS IoT Greengrass V1，連線的裝置是您新增至 Greengrass 群組以連線至該群組中核心裝置並透過 MQTT 通訊的 AWS IoT 物件。每次新增或移除連線的裝置時，您必須部署該群組。您可以使用訂閱，在已連線的裝置 AWS IoT Core和核心裝置上的應用程式之間轉送訊息。	在 中 AWS IoT Greengrass V2，連線的裝置稱為 Greengrass 用戶端裝置。 - 您可以將用戶端裝置與核心裝置建立關聯，以連接它們並透過 MQTT 進行通訊。 - 若要授權用戶端裝置進行連線，您可以定義可套用至用戶端裝置群組的授權政策，因此您不需要建立部署來新增或移除用戶端裝置。 - 若要在用戶端裝置 AWS IoT Core和 Greengrass 元件之間轉送訊息，您可以設定選用的 MQTT 橋接元件。 在 AWS IoT Greengrass V1 和 中 AWS IoT Greengrass V2，裝置可以執行 FreeRTOS，或使用 AWS IoT Device SDK或 Greengrass 探索 API 來取得其可連線之核心裝置的相關資訊。Greengrass 探索 API 可回溯相容，因此，如果您有用戶端裝置連接到 V1 核心裝置，您可以將它們連接到 V2 核心裝置，而無需變更其程式碼。 如需用戶端裝置的詳細資訊，請參閱 與本機 IoT 裝置互動。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
本機資源	在 中 AWS IoT Greengrass V1，可在容器中執行的 Lambda 函數可設定為存取核心裝置檔案系統的磁碟區和裝置。這些檔案系統資源稱為本機資源。	在 中 AWS IoT Greengrass V2，您可以執行 Lambda 函數、Docker 容器或原生化作業系統程序或自訂執行時間的元件。 - 當您匯入容器化 Lambda 函數做為元件時，您必須指定函數使用的本機資源。 - 非容器化 Lambda 函數和非 Lambda 元件可以直接與核心裝置上的本機資源搭配使用，因此您不需要指定元件使用的本機資源。
本機影子服務	在 中 AWS IoT Greengrass V1，本機影子服務預設為啟用，並僅支援未命名的傳統影子。您可以在 Lambda 函數中使用 AWS IoT Greengrass 核心 SDK 來與裝置上的陰影互動。	在 中 AWS IoT Greengrass V2，您可以透過部署影子管理員元件來啟用本機影子服務。 - 您可以使用 Lambda 函數中的 AWS IoT Device SDK V2 和自訂元件，與裝置上的陰影互動。 - 本機影子服務支援具名影子。 - 本機影子服務可讓您刪除影子，並使用 同步已刪除的影子 AWS IoT Core。 如需詳細資訊，請參閱與裝置陰影互動。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
訂閱	在 中 AWS IoT Greengrass V1，您可以定義 Greengrass 群組的訂閱，以指定 Lambda 函數、連接器、連線裝置、AWS IoT Core MQTT 代理程式和本機影子服務之間的通訊通道。訂閱會指定 Lambda 函數接收事件訊息的位置，以做為函數承載使用。	在 中 AWS IoT Greengrass V2，您可以指定通訊管道，而無需使用訂閱。 • 元件會管理自己的通訊管道，以與本機發佈/訂閱訊息、AWS IoT Core MQTT 訊息和本機影子服務互動。 • 若要開發對來自其他元件或 AWS IoT Core MQTT 代理程式的訊息做出反應的元件，您可以使用處理間通訊 (IPC) 介面進行本機發佈/訂閱訊息和 AWS IoT Core MQTT 訊息。 • 若要開發與本機影子服務互動的元件，您可以使用本機影子服務的 IPC 介面。 • 在元件組態中，您可以定義授權政策，以指定元件有權使用的主題和本機影子。 • 若要設定用戶端裝置、本機發佈/訂閱代理程式和 AWS IoT Core MQTT 代理程式之間的通訊通道，您可以設定和部署 MQTT 橋接元件。MQTT 橋接元件可讓您與元件中的用戶端裝置互動，並在用戶端裝置與 之間轉送訊息 AWS IoT Core。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
存取其他 AWS 服務	在 中 AWS IoT Greengrass V1，您將稱為 群組角色的 AWS Identity and Access Management (IAM) 角色連接至 Greengrass 群組。群組角色定義 Lambda 在該群組的核心裝置上用於存取的 函數和 AWS IoT Greengrass 功能許可 AWS 服務。	在 中 AWS IoT Greengrass V2，您將 AWS IoT 角色別名連接至 Greengrass 核心裝置。角色別名指向稱為字符交換角色的 IAM 角色。權杖交換角色會定義核心裝置上 Greengrass 元件用來存取的許可 AWS 服務。如需詳細資訊，請參閱 授權核心裝置與 AWS 服務互動 。

選擇您的執行時間 (Greengrass nucleus 或 Greengrass nucleus lite)

在 Greengrass 核和 Greengrass 核精簡之間進行選擇取決於您的裝置資源和 Lambda 函數使用的功能。檢閱下表中的事件來源相容性矩陣，然後使用決策流程圖來判斷適合您遷移的執行時間。如需 Greengrass 核和 Greengrass 核精簡功能的詳細比較，請參閱[選擇您的執行時間](#)。

事件來源相容性矩陣

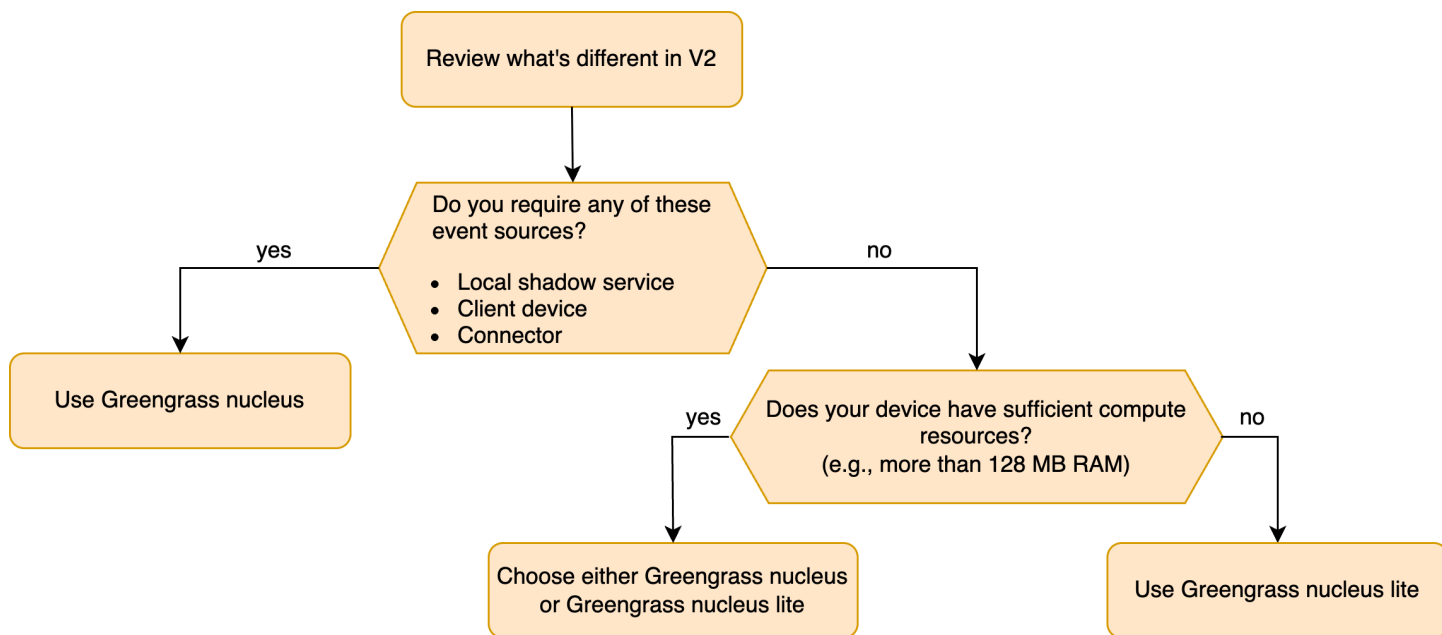
在 中 AWS IoT Greengrass V1，Lambda 函數可以與五種類型的事件來源通訊：其他 Lambda 函數、AWS IoT Core、本機影子服務、用戶端裝置和連接器。下表顯示每個 V2 執行時間支援哪些事件來源。

注意：事件來源名稱使用 AWS IoT Greengrass V1 術語。遷移至 V2 時，Lambda 函數會轉換為 Lambda 元件（僅支援 Greengrass 核）或一般元件（同時支援 Greengrass 核和 Greengrass 核精簡型）。

事件來源	Greengrass 核	Greengrass nucleus lite
群組中的其他 Lambda 函數	✓ (Lambda 元件和一般元件)	✓ (僅限通用元件)
AWS IoT Core 服務	✓	✓

事件來源	Greengrass 核	Greengrass nucleus lite
本機陰影服務	✓	✗
用戶端裝置	✓	✗
連接器	✓	✗

執行時間選擇決策流程



備註

- 如需 Greengrass nucleus lite 需求和相容性詳細資訊，請參閱 [Greengrass nucleus lite](#)。Greengrass nucleus lite 至少需要 5 MB RAM，專為資源受限的裝置而設計。
- 決策流程會根據一般使用案例提供指引，但不是嚴格的要求。同時具有資源限制和資源足夠的裝置的客戶，可以選擇在所有裝置上使用單一執行時間，以簡化操作，即使有些裝置可以支援任一執行時間。

後續步驟

選擇執行時間後，請繼續設定您的測試裝置：

- 針對 Greengrass 核執行時間：[設定新裝置以在 V2 上測試 V1 應用程式 V2](#)
- 針對 Greengrass nucleus lite 執行時間：[設定具有 Greengrass nucleus lite 的新裝置](#)

設定新裝置以測試 V2 上的 V1 應用程式 V2

若要將生產環境中裝置的風險降至最低，請在升級生產裝置之前，建立新的裝置以在 V2 上測試 V1 應用程式。

根據您在上一個步驟中的執行時間選擇，選擇下列其中一個設定指南：

- 選項 A - Greengrass 核執行時間：[設定新裝置以測試 V2 上的 V1 應用程式 V2](#)如果您選擇 Greengrass 核，請遵循。使用此選項，您可以將 Lambda 函數匯入為 Lambda 元件，且程式碼變更最少，並支援本機影子服務、用戶端裝置和連接器等 V1 功能。
- 選項 B - Greengrass nucleus lite 執行時間：[設定新裝置以測試 V2 上的 V1 應用程式 \(Greengrass nucleus lite\) V2](#)如果您選擇 Greengrass nucleus lite，請遵循。此選項需要使用 AWS IoT Device SDK V2 或元件 SDK 將 Lambda 函數轉換為一般 AWS IoT Greengrass 元件，但針對資源受限的裝置進行最佳化。

設定新裝置以測試 V2 上的 V1 應用程式 V2

設定新的 AWS IoT Greengrass V2 核心裝置，為您的 AWS IoT Greengrass V1 應用程式部署和測試 AWS 提供的元件和 AWS Lambda 函數。您也可以使用此 V2 核心裝置來開發和測試在核心裝置上執行原生程序的其他自訂 Greengrass 元件。在 V2 核心裝置上測試應用程式後，您可以將現有的 V1 核心裝置升級至 V2，並部署提供 V1 功能的 V2 元件。

步驟 1：AWS IoT Greengrass V2 在新裝置上安裝

在新裝置上安裝 AWS IoT Greengrass Core 軟體 v2.x。您可以遵循[入門教學](#)課程來設定裝置，並了解如何開發和部署元件。本教學課程使用[自動佈建](#)來快速設定裝置。安裝 AWS IoT Greengrass Core 軟體 v2.x 時，請指定 `--deploy-dev-tools` 引數來部署 [Greengrass CLI](#)，讓您可以直接在裝置上開發、測試和偵錯元件。如需其他安裝選項的詳細資訊，包括如何在代理後方安裝 AWS IoT Greengrass Core 軟體或使用硬體安全模組 (HSM)，請參閱 [安裝 AWS IoT Greengrass Core 軟體](#)。

(選用) 啟用記錄至 Amazon CloudWatch Logs

若要讓 V2 核心裝置將日誌上傳至 Amazon CloudWatch Logs，您可以部署 AWS 提供的 [日誌管理員元件](#)。您可以使用 CloudWatch Logs 來檢視元件日誌，因此您可以在無法存取核心裝置的檔案系統的情況下進行偵錯和疑難排解。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

步驟 2：建立和部署 AWS IoT Greengrass V2 元件以遷移 AWS IoT Greengrass V1 應用程式

您可以在上執行大多數 AWS IoT Greengrass V1 應用程式 AWS IoT Greengrass V2。您可以將 Lambda 函數匯入為上執行的元件 AWS IoT Greengrass V2，而且您可以使用 [AWS 提供的元件](#)，提供與 AWS IoT Greengrass 連接器相同的功能。

您也可以開發自訂元件，以建置要在 Greengrass 核心裝置上執行的任何功能或執行時間。如需有關如何在本機開發和測試元件的資訊，請參閱 [建立 AWS IoT Greengrass 元件](#)。

主題

- [匯入 V1 Lambda 函數](#)
- [使用 V1 連接器](#)
- [執行 Docker 容器](#)
- [連接 V1 Greengrass 裝置](#)
- [啟用本機影子服務](#)
- [與整合 AWS IoT SiteWise](#)

匯入 V1 Lambda 函數

您可以匯入 Lambda 函數做為 AWS IoT Greengrass V2 元件。從下列方法中選擇：

- 將 V1 Lambda 函數直接匯入為 Greengrass 元件。
- 更新您的 Lambda 函數以使用 AWS IoT Device SDK v2 中的 Greengrass 程式庫，然後將 Lambda 函數匯入為 Greengrass 元件。
- 建立使用非 Lambda 程式碼和 AWS IoT Device SDK v2 的自訂元件，以實作與 Lambda 函數相同的功能。

如果您的 Lambda 函數使用功能，例如串流管理員或本機秘密，則必須在封裝這些功能的 AWS 提供的元件上定義相依性。當您部署 Lambda 函數元件時，部署也會包含您定義為相依性之每個功能的

元件。在部署中，您可以設定參數，例如要部署到核心裝置的秘密。並非所有 V1 功能都需要 V2 上 Lambda 函數的元件相依性。下列清單說明如何在 V1/V2 功能：

- 存取其他 AWS 服務

如果您的 Lambda 函數使用 AWS 登入資料向其他服務 AWS 提出請求，則核心裝置的字符交換角色必須允許核心裝置執行 Lambda 函數使用 AWS 的操作。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

- 串流管理員

如果您的 Lambda 函數使用串流管理員，請在匯入函數時指定 `aws.greengrass.StreamManager` 做為元件相依性。部署串流管理員元件時，請指定要為目標核心裝置設定的串流管理員參數。核心裝置的字符交換角色必須允許核心裝置存取您搭配串流管理員使用的 AWS 雲端目的地。如需詳細資訊，請參閱[串流管理員](#)。

- 本機秘密

如果您的 Lambda 函數使用本機秘密，請在匯入函數時指定 `aws.greengrass.SecretManager` 做為元件相依性。當您部署秘密管理員元件時，請指定要部署到目標核心裝置的秘密資源。核心裝置的字符交換角色必須允許核心裝置擷取要部署的秘密資源。如需詳細資訊，請參閱[秘密管理員](#)。

部署 Lambda 函數元件時，請將其設定為具有 [IPC 授權政策](#)，授予在 AWS IoT Device SDK V2 中使用 [GetSecretValue IPC 操作](#) 的許可。

- 本機陰影

如果您的 Lambda 函數與本機影子互動，您必須更新 Lambda 函數程式碼才能使用 AWS IoT Device SDK V2。匯入函數時，您也必須將指定 `aws.greengrass.ShadowManager` 為元件相依性。如需詳細資訊，請參閱[與裝置陰影互動](#)。

部署 Lambda 函數元件時，請將其設定為具有 [IPC 授權政策](#)，授予在 AWS IoT Device SDK V2 中使用 [陰影 IPC 操作](#) 的許可。

- 訂閱

- 如果您的 Lambda 函數從雲端來源訂閱訊息，請在匯入函數時將這些訂閱指定為事件來源。
- 如果您的 Lambda 函數從另一個 Lambda 函數訂閱訊息，或者您的 Lambda 函數將訊息發佈至 AWS IoT Core 或其他 Lambda 函數，請在部署 Lambda 函數時設定和部署[舊版訂閱路由器元件](#)。部署舊版訂閱路由器元件時，請指定 Lambda 函數使用的訂閱。

 Note

只有在您的 Lambda 函數在 AWS IoT Greengrass 核心 SDK 中使用 `publish()` 函數時，才需要舊版訂閱路由器元件。如果您更新 Lambda 函數程式碼以使用 AWS IoT Device SDK V2 中的程序間通訊 (IPC) 介面，則不需要部署舊版訂閱路由器元件。如需詳細資訊，請參閱下列[程序間通訊服務](#)：

- [發佈/訂閱本機訊息](#)
- [發佈/訂閱 AWS IoT Core MQTT 訊息](#)

- 如果您的 Lambda 函數訂閱來自本機連線裝置的訊息，請在匯入函數時將這些訂閱指定為事件來源。您還必須設定和部署 [MQTT 橋接器元件](#)，將訊息從連接的裝置轉送到您指定為事件來源的本機發佈/訂閱主題。
- 如果您的 Lambda 函數發佈訊息到本機連線的裝置，您必須更新 Lambda 函數程式碼，以使用 AWS IoT Device SDK V2 [發佈本機發佈/訂閱訊息](#)。您也必須設定和部署 [MQTT 橋接器元件](#)，將訊息從本機發佈/訂閱訊息代理程式轉送至連線的裝置。
- 本機磁碟區和裝置

如果您的容器化 Lambda 函數存取本機磁碟區或裝置，請在匯入 Lambda 函數時指定這些磁碟區和裝置。此功能不需要元件相依性。

如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

使用 V1 連接器

您可以部署 AWS 提供的元件，這些元件提供與某些 AWS IoT Greengrass 連接器相同的功能。建立部署時，您可以設定連接器的參數。

下列 AWS IoT Greengrass V2 元件提供 Greengrass V1 連接器功能：

- [CloudWatch 指標元件](#)
- [AWS IoT Device Defender 元件](#)
- [Firehose 元件](#)
- [Modbus-RTU 通訊協定轉接器元件](#)
- [Amazon SNS 元件](#)

執行 Docker 容器

AWS IoT Greengrass V2 不提供元件來直接取代 V1 Docker 應用程式部署連接器。不過，您可以使用 Docker 應用程式管理員元件下載 Docker 映像，然後建立從下載的映像執行 Docker 容器的自訂元件。如需詳細資訊，請參閱[執行 Docker 容器](#)及[Docker 應用程式管理員](#)。

連接 V1 Greengrass 裝置

在中連接的裝置 AWS IoT Greengrass V1 稱為用戶端裝置 in AWS IoT Greengrass V2。用戶端裝置的 AWS IoT Greengrass V2 支援與回溯相容 AWS IoT Greengrass V1，因此您可以將 V1 用戶端裝置連線到 V2 核心裝置，而無需變更其應用程式碼。若要讓用戶端裝置連線至 V2 核心裝置，請部署啟用用戶端裝置支援的 Greengrass 元件，並將用戶端裝置與核心裝置建立關聯。若要在 AWS IoT Core 用戶端裝置、雲端服務和 Greengrass 元件（包括 Lambda 函數）之間轉送訊息，請部署和設定 [MQTT 橋接器元件](#)。您可以部署 [IP 偵測器元件](#)以自動偵測連線資訊，也可以手動管理端點。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

啟用本機影子服務

在中 AWS IoT Greengrass V2，本機影子服務是由 AWS 提供的影子管理員元件實作。AWS IoT Greengrass V2 也包含對具名影子的支援。若要讓您的元件與本機影子互動，並將影子狀態同步至 AWS IoT Core，請設定和部署影子管理員元件，並在元件程式碼中使用影子 IPC 操作。如需詳細資訊，請參閱[與裝置陰影互動](#)。

與整合 AWS IoT SiteWise

如果您使用 V1 核心裝置做為 AWS IoT SiteWise 閘道，[請依照指示](#)將新的 V2 核心裝置設定為 AWS IoT SiteWise 閘道。AWS IoT SiteWise 提供可為您部署 AWS IoT SiteWise 元件的安裝指令碼。

步驟 3：測試您的 AWS IoT Greengrass V2 應用程式

建立 V2 元件並將其部署到新的 V2 核心裝置之後，請確認您的應用程式符合您的期望。您可以檢查裝置的日誌，以檢視元件的標準輸出 (stdout) 和標準錯誤 (stderr) 訊息。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如果您將 [Greengrass CLI](#) 部署到核心裝置，您可以使用它來偵錯元件及其組態。如需詳細資訊，請參閱[Greengrass CLI 命令](#)。

確認應用程式可在 V2 核心裝置上運作後，您可以將應用程式的 Greengrass 元件部署至其他核心裝置。如果您開發了執行原生程序或 Docker 容器的自訂元件，您必須先將這些[元件發佈](#)到 AWS IoT Greengrass 服務，以將其部署到其他核心裝置。

設定新裝置以測試 V2 上的 V1 應用程式 (Greengrass nucleus lite) V2

使用 Greengrass nucleus lite 設定新裝置，以測試您建立的一般元件，將 AWS IoT Greengrass V1 應用程式遷移至 V2。Greengrass nucleus lite 是一種輕量型執行時間，針對資源受限的裝置進行最佳化。您可以使用此裝置來開發和測試執行原生程序的自訂 Greengrass 元件。在 Greengrass nucleus lite 裝置上測試應用程式後，您可以將 V2 元件部署到執行 Greengrass nucleus lite 的其他裝置，或將現有的 V1 核心裝置升級至 V2。

步驟 1：在新裝置上安裝 Greengrass nucleus lite

在新裝置上安裝 Greengrass nucleus lite。遵循 [Greengrass nucleus lite 安裝指南](#) 來設定裝置。

Note

Greengrass nucleus lite 現在不支援本機影子服務、用戶端裝置或連接器。在繼續本指南之前，請確定您的 V1 應用程式不依賴這些功能。

步驟 2：建立和部署一般元件以遷移 AWS IoT Greengrass V1 Lambda 函數

若要在 Greengrass nucleus lite 上複寫 AWS IoT Greengrass V1 Lambda 函數的功能，您必須將其轉換為一般元件。這包括重寫您的 Lambda 函數程式碼，以使用 AWS IoT Device SDK V2 或 AWS IoT Greengrass 元件 SDK，而非 AWS IoT Greengrass 核心 SDK。

下表列出本指南中 V2 元件範例中所使用的 SDKs：

SDK	最低版本
AWS IoT Device SDK for Python v2	1.11.3 版
AWS IoT Device SDK 適用於 Java 的 v2	1.9.3 版
AWS IoT Device SDK for JavaScript v2	v1.12.0
AWS IoT Greengrass 元件 SDK (C/C++) (目前為預覽版)	0.4.0 版

以下範例顯示使用 AWS IoT Greengrass V1 Core SDK 及其同等一般元件的 Lambda 函數，其中包含兩種主要案例的多個程式設計語言的元件程式碼、配方和建置說明：

- 本機通訊 - 使用本機 pub/sub 與相同裝置上的其他元件通訊的元件
- 雲端通訊 - 與 AWS IoT Core 或其他 AWS 服務通訊的元件

案例 1：本機通訊（發佈者 → 處理器 → 訂閱者）

此案例示範將使用本機 pub/sub 通訊的 V1 Lambda 函數轉換為 V2 一般元件。

應用程式架構

此範例包含三個元件：

- 發佈者 Lambda 發佈溫度資料
- 處理器 Lambda 接收和處理資料
- 處理器 Lambda 會將處理的結果發佈給訂閱者 Lambda

下面的程式碼範例著重於處理器 Lambda，這示範了訂閱和發佈訊息以進行本機通訊。

V1 群組訂閱

在中 AWS IoT Greengrass V1，下列群組訂閱會啟用 Lambda 函數之間的通訊：

訂閱 1：發佈者 → 處理器

- 來源：Lambda（發佈者）
- 目標：Lambda（處理器）
- 主題：感應器/溫度

訂閱 2：處理器 → 訂閱者

- 來源：Lambda（處理器）
- 目標：Lambda（訂閱者）
- 主題：lambda/alerts

處理器 Lambda 函數 (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 = strdup((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 << "°F" << 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;
}

```

一般元件 (V2)

若要在 中實現相同的功能 AWS IoT Greengrass V2，請使用下列項目建立一般元件：

1. 元件程式碼

Python

先決條件：使用此元件程式碼之前，請在 Greengrass 裝置上安裝並驗證 AWS IoT Device SDK 適用於 Python 的：

```
# Install the SDK
pip3 install awsiotsdk

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

如果您在安裝期間遇到相依性衝突，請嘗試安裝特定版本的 AWS IoT Device SDK。

如果驗證命令列印「SDK 安裝成功」，您就可以使用元件程式碼：

```
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. 建置和封裝元件

有些語言需要在部署之前建置或封裝。

Python

Python 不需要編譯。元件可以直接使用 .py 檔案。

Java

若要建置綁定所有相依性的可執行 JAR：

1. 在專案目錄中建立pom.xml檔案：

```
<?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. 建置 JAR :

```
mvn clean package
```

這會target/temperature-processor-1.0.0.jar建立包含所有相依性的。

3. 將 JAR 上傳至 S3 儲存貯體以進行部署。

JavaScript

若要使用所有相依性封裝 Node.js 元件：

1. 建立package.json檔案：

```
{
  "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. 在開發機器上安裝相依性：

```
npm install
```

這會建立包含 AWS IoT Device SDK v2 的 `node_modules` 資料夾。

3. 用於部署的套件：

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

4. 將 zip 檔案上傳至 S3 儲存貯體以進行部署。

 Note

Greengrass 裝置必須安裝 Node.js 執行期 (14 版或更新版本)。您不需要 `npm install` 在 Greengrass 核心裝置上執行，因為元件成品包含綁定 `node_modules` 資料夾中的所有相依性。

C

先決條件：

若要建置 SDK 和 元件，您需要下列建置相依性：

- GCC 或 Clang
- CMake (至少 3.22 版)
- Make 或 Ninja

安裝建置相依性：

在 Ubuntu/Debian 上：

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

在 Amazon Linux 上：

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

為您的元件建立 `CMakeLists.txt` 檔案：

```
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)
```

建置步驟：

```
# 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++

先決條件：

若要建置 SDK 和 元件，您需要下列建置相依性：

- 支援 C++20 的 GCC 或 Clang
- CMake (至少 3.22 版)
- Make 或 Ninja

安裝建置相依性：

在 Ubuntu/Debian 上：

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

在 Amazon Linux 上：

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

為您的元件建立 CMakeLists.txt 檔案：

```
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++)
```

建置步驟：

```
# 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. 元件配方

使用您元件使用的實際主題來更新「資源」陣列。

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"
        }
      ]
    }
  ]
}

```

案例 2：雲端通訊

此案例示範將與 通訊的 V1 Lambda 函數轉換為 V2 AWS IoT Core 一般元件。

應用程式架構

此範例使用雲端連線架構：

- AWS IoT Core 將命令傳送至裝置
- 控制器 Lambda 接收並處理命令
- 控制器 Lambda 將遙測資料傳回至 AWS IoT Core

此範例著重於控制器 Lambda，示範從 接收訊息以及向其發佈訊息 AWS IoT Core。

V1 群組訂閱

在 中 AWS IoT Greengrass V1，下列群組訂閱可啟用 Lambda 函數與 之間的通訊 AWS IoT Core：

訂閱 1：IoT 雲端 → Lambda

- 來源：IoT Cloud
- 目標：Lambda (DeviceController)
- 主題：命令/裝置 1

訂閱 2：Lambda → IoT 雲端

- 來源：Lambda (DeviceController)
- 目標：IoT 雲端
- 主題：遙測/裝置1

控制器 Lambda 函數 (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 = strdup((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;
}

```

一般元件 (V2)

若要在 中實現相同的功能 AWS IoT Greengrass V2，請使用下列項目建立一般元件：

1. 元件程式碼

Python

先決條件：使用此元件程式碼之前，請在 Greengrass 裝置上安裝並驗證 AWS IoT Device SDK 適用於 Python 的：

```

# Install the SDK
pip3 install awsiotsdk

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

如果您在安裝期間遇到相依性衝突，請嘗試安裝特定版本的 AWS IoT Device SDK。

如果驗證命令列印「SDK 安裝成功」，您就可以使用元件程式碼：

```
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. 建置和封裝元件

有些語言需要在部署之前建置或封裝。

Python

Python 不需要編譯。元件可以直接使用 .py 檔案。

Java

若要建置綁定所有相依性的可執行 JAR：

1. 在專案目錄中建立 pom.xml 檔案：

```

<?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. 建置 JAR :

```
mvn clean package
```

這會target/device-controller-1.0.0.jar建立包含所有相依性的。

3. 將 JAR 上傳至 S3 儲存貯體以進行部署。

JavaScript

若要使用所有相依性封裝 Node.js 元件：

1. 建立package.json檔案：

```
{
  "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. 在開發機器上安裝相依性：

```
npm install
```

這會建立包含 AWS AWS IoT Device SDK v2 的node_modules資料夾。

3. 用於部署的套件：

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

4. 將 zip 檔案上傳至 S3 儲存貯體以進行部署。

Note

Greengrass 裝置必須安裝 Node.js 執行期 (14 版或更新版本)。您不需要npm install在 Greengrass 核心裝置上執行，因為元件成品包含綁定node_modules資料夾中的所有相依性。

C

先決條件：

若要建置 SDK 和 元件，您需要下列建置相依性：

- GCC 或 Clang
- CMake (至少 3.22 版)
- Make 或 Ninja

安裝建置相依性：

在 Ubuntu/Debian 上：

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

在 Amazon Linux 上：

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

為您的元件建立 CMakeLists.txt 檔案：

```
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)
```

建置步驟：

```
# 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++

先決條件：

若要建置 SDK 和 元件，您需要下列建置相依性：

- 支援 C++20 的 GCC 或 Clang
- CMake (至少 3.22 版)
- Make 或 Ninja

安裝建置相依性：

在 Ubuntu/Debian 上：

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

在 Amazon Linux 上：

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

為您的元件建立 CMakeLists.txt 檔案：

```
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++)
```

建置步驟：

```
# 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. 元件配方

使用您元件使用的實際主題來更新「資源」陣列。

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"  
      }  
    ]  
  }  
]  
}
```

升級 Greengrass V1 核心裝置至 Greengrass V2

確認您的應用程式和元件可在 AWS IoT Greengrass V2 核心裝置上運作後，您可以在目前執行 v1.x 的裝置上安裝 AWS IoT Greengrass Core 軟體 v2.x，例如生產裝置。然後，部署 Greengrass V2 元件以在裝置上執行 Greengrass 應用程式。

若要將裝置機群從 V1 升級到 V2，請完成以下步驟，讓每個裝置升級。您可以使用物件群組將 V2 元件部署至核心裝置機群。

Tip

我們建議您建立指令碼，以自動化裝置機群的升級程序。如果您使用 [AWS Systems Manager](#) 管理機群，您可以使用 Systems Manager 在每個裝置上執行該指令碼，將機群從 V1 升級到 V2。

您可以聯絡您的 AWS 企業支援代表，詢問如何以最佳方式自動化升級程序。

步驟 1：安裝 AWS IoT Greengrass Core 軟體 v2.x

從下列選項中選擇，在 V1 核心裝置上安裝 AWS IoT Greengrass Core 軟體 v2.x：

- [在較少的步驟中升級](#)

若要在較少的步驟中升級，您可以在安裝 v2.x 軟體之前解除安裝 v1.x 軟體。

- [以最少的停機時間進行升級](#)

若要以最短的停機時間進行升級，您可以同時安裝兩個版本的 AWS IoT Greengrass Core 軟體。安裝 AWS IoT Greengrass Core 軟體 v2.x 並確認 V2 應用程式是否正常運作後，請解除安裝 AWS IoT Greengrass Core 軟體 v1.x。選擇此選項之前，請考慮同時執行兩個 AWS IoT Greengrass Core 軟體版本所需的額外 RAM。

在安裝 v2.x 之前解除安裝 AWS IoT Greengrass Core v1.x

如果您想要循序升級，請在裝置上安裝 v2.x 之前解除安裝 AWS IoT Greengrass Core 軟體 v1.x。

解除安裝 AWS IoT Greengrass Core 軟體 v1.x

1. 如果 AWS IoT Greengrass Core 軟體 v1.x 以服務的形式執行，您必須停止、停用和移除服務。

- a. 停止執行中的 AWS IoT Greengrass Core 軟體 v1.x 服務。

```
sudo systemctl stop greengrass
```

- b. 等待服務停止。您可以使用 `list` 命令來檢查服務的狀態。

```
sudo systemctl list-units --type=service | grep greengrass
```

- c. 停用服務。

```
sudo systemctl disable greengrass
```

- d. 移除 服務。

```
sudo rm /etc/systemd/system/greengrass.service
```

2. 如果 AWS IoT Greengrass Core 軟體 v1.x 不是以服務的形式執行，請使用下列命令來停止協助程式。以 *Greengrass ##### greengrass-root*。預設位置為 `/greengrass`。

```
cd /greengrass-root/ggc/core/  
sudo ./greengrassd stop
```

3. (選用) 將 Greengrass 根資料夾和適用的[自訂寫入資料夾](#)備份到裝置上的不同資料夾。

- a. 使用以下命令將目前的 Greengrass 根資料夾複製到不同的資料夾，然後移除根資料夾。

```
sudo cp -r /greengrass-root /path/to/greengrass-backup  
rm -rf /greengrass-root
```

- b. 使用以下命令將寫入資料夾移至不同的資料夾，然後移除寫入資料夾。

```
sudo cp -r /write-directory /path/to/write-directory-backup  
rm -rf /write-directory
```

針對 Greengrass 核：您可以使用 [安裝說明 AWS IoT Greengrass V2](#)，在裝置上安裝 Greengrass 核。

針對 Greengrass nucleus lite：您可以使用 [Greengrass nucleus lite 的安裝指示](#)來安裝 Greengrass nucleus lite。

 Tip

若要在從 V1 遷移至 V2 時重複使用核心裝置的身分，請依照指示[使用手動佈建安裝 AWS IoT Greengrass 核心軟體](#)。首先從裝置移除 V1 核心軟體，然後重複使用 V1 核心裝置的 AWS IoT 物件和憑證，並更新憑證 AWS IoT 的政策以授予 v2.x 軟體所需的許可。

在已執行 v1.x 的裝置上安裝 AWS IoT Greengrass Core 軟體 v2.x

如果您在已執行 AWS IoT Greengrass Core 軟體 v1.x 的裝置上安裝 AWS IoT Greengrass Core v2.x 軟體，請記住下列事項：

- V2 核心裝置的 AWS IoT 物件名稱必須是唯一的。請勿使用與 V1 核心裝置相同的物件名稱。
- 您用於 AWS IoT Greengrass Core 軟體 v2.x 的連接埠必須與您用於 v1.x 的連接埠不同。
 - 將 V1 串流管理員設定為使用 8088 以外的連接埠。如需詳細資訊，請參閱[設定串流管理員](#)。
 - 將 V1 MQTT 代理程式設定為使用 8883 以外的連接埠。如需詳細資訊，請參閱[設定本機傳訊的 MQTT 連接埠](#)。
- AWS IoT Greengrass V2 不提供重新命名 Greengrass 系統服務的選項。如果您執行 Greengrass 做為系統服務，則必須執行下列其中一項操作，以避免系統服務名稱衝突：
 - 在安裝 v2.x 之前，重新命名適用於 v1.x 的 Greengrass 服務。
 - 在沒有系統服務的情況下安裝 AWS IoT Greengrass Core 軟體 v2.x，然後手動[將軟體設定為名為 以外的系統服務](#)greengrass。

重新命名 v1.x 的 Greengrass 服務

1. 停止 AWS IoT Greengrass Core 軟體 v1.x 服務。

```
sudo systemctl stop greengrass
```

2. 等待服務停止。服務最多可能需要幾分鐘的時間才能停止。您可以使用 `list-units` 命令來檢查服務是否停止。

```
sudo systemctl list-units --type=service | grep greengrass
```

3. 停用服務。

```
sudo systemctl disable greengrass
```

4. 重新命名服務。

```
sudo mv /etc/systemd/system/greengrass.service /etc/systemd/system/greengrass-v1.service
```

5. 重新載入服務並啟動服務。

```
sudo systemctl daemon-reload
sudo systemctl reset-failed
sudo systemctl enable greengrass-v1
sudo systemctl start greengrass-v1
```

然後，您可以使用 [的安裝指示 AWS IoT Greengrass V2](#)，在裝置上安裝軟體。

 Tip

若要在從 V1 遷移至 V2 時重複使用核心裝置的身分，請依照指示[使用手動佈建安裝 AWS IoT Greengrass 核心軟體](#)。首先從裝置移除 V1 核心軟體，然後重複使用 V1 核心裝置的 AWS IoT 物件和憑證，並更新憑證 AWS IoT 的政策以授予 v2.x 軟體所需的許可。

步驟 2：將 AWS IoT Greengrass V2 元件部署至核心裝置

在裝置上安裝 AWS IoT Greengrass Core 軟體 v2.x 之後，請根據您選擇的執行時間部署元件。

針對 Greengrass 核：

建立包含下列資源的部署。若要將元件部署到類似裝置的機群，請為包含這些裝置的物件群組建立部署。

- 您從 V1 Lambda 函數建立的 Lambda 函數元件。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。
- 如果您使用 V1 訂閱，則為[舊版訂閱路由器元件](#)。

- 如果您使用串流管理員，則為[串流管理員元件](#)。如需詳細資訊，請參閱[管理 Greengrass 核心裝置上的資料串流](#)。
- 如果您使用本機秘密，則為[秘密管理員元件](#)。
- 如果您使用 V1 連接器，則為 [AWS 提供的連接器元件](#)。
- 如果您使用 Docker 容器，則為 [Docker 應用程式管理員元件](#)。如需詳細資訊，請參閱[執行 Docker 容器](#)。
- 如果您使用連線的裝置，則為[用戶端裝置支援的元件](#)。您還必須啟用用戶端裝置支援，並將用戶端裝置與您的核心裝置建立關聯。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。
- 如果您使用裝置影子，則為[影子管理員元件](#)。如需詳細資訊，請參閱[與裝置陰影互動](#)。
- 如果您將日誌從 Greengrass 核心裝置上傳至 Amazon CloudWatch Logs，即[日誌管理員元件](#)。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。
- 如果您與 整合 AWS IoT SiteWise，[請依照指示](#)將 V2 核心裝置設定為 AWS IoT SiteWise 閘道。AWS IoT SiteWise 提供為您部署 AWS IoT SiteWise 元件的安裝指令碼。
- 您開發來實作自訂功能的使用者定義元件。

如需建立和修訂部署的資訊，請參閱 [將 AWS IoT Greengrass 元件部署至裝置](#)。

針對 Greengrass nucleus lite：

將您在[遷移指南的步驟 2](#)中建立的一般元件部署到 Greengrass nucleus lite 裝置：

1. 使用您建立的元件配方在 中建立元件
2. 建立以 Greengrass nucleus lite 裝置為目標的部署，其中包含您的一般元件
3. 確認您的元件是否正確執行

教學課程：入門 AWS IoT Greengrass V2

您可以完成此入門教學課程，了解的基本功能 AWS IoT Greengrass V2。在此教學課程中，您將執行下列操作：

1. 在 Linux 裝置上安裝和設定 AWS IoT Greengrass Core 軟體，例如 Raspberry Pi 或 Windows 裝置。此裝置是 Greengrass 核心裝置。
2. 在 Greengrass 核心裝置上開發 Hello World 元件。元件是在 Greengrass 核心裝置上執行的軟體模組。
3. 在 AWS IoT Greengrass V2 中將該元件上傳至 AWS 雲端。
4. 將該元件從 部署 AWS 雲端 到您的 Greengrass 核心裝置。

Note

本教學說明如何設定開發環境並探索的功能 AWS IoT Greengrass。如需如何設定和設定生產裝置的詳細資訊，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心裝置](#)
- [安裝 AWS IoT Greengrass Core 軟體](#)

您預計將花費 20 到 30 分鐘在本教學課程中。

主題

- [先決條件](#)
- [步驟 1：設定 AWS 帳戶](#)
- [步驟 2：設定您的環境](#)
- [步驟 3：安裝 AWS IoT Greengrass 核心軟體](#)
- [步驟 4：在您的裝置上開發和測試元件](#)
- [步驟 5：在 AWS IoT Greengrass 服務中建立元件](#)
- [步驟 6：部署您的元件](#)
- [後續步驟](#)

先決條件

若要完成此入門教學課程，您需要以下項目：

- AWS 帳戶：如果您沒有帳戶，請參閱 [步驟 1：設定 AWS 帳戶](#)。
- 使用 [AWS 區域](#) 支援的 AWS IoT Greengrass V2。如需支援區域的清單，請參閱《AWS 一般參考》中的 [AWS IoT Greengrass V2 端點和配額](#)。
- 具有管理員許可的 AWS Identity and Access Management (IAM) 使用者。
- 設定為 Greengrass 核心裝置的裝置，例如 Raspberry Pi 搭配 [Raspberry Pi 作業系統](#)（先前稱為 Raspbian）或 Windows 10 裝置。您必須擁有此裝置的管理員許可，或能夠取得管理員權限，例如透過 sudo。此裝置必須具有網際網路連線。

您也可以選擇使用符合安裝和執行 AWS IoT Greengrass Core 軟體需求的不同裝置。

如果您的開發電腦符合這些要求，您可以在本教學課程中將其設定為 Greengrass 核心裝置。

- 裝置上所有使用者已安裝 [Python](#) 3.5 或更新版本，並已新增至 PATH 環境變數。在 Windows 上，您也必須為所有使用者安裝適用於 Windows 的 Python Launcher。

Important

在 Windows 中，Python 預設不會為所有使用者安裝。安裝 Python 時，您必須自訂安裝以設定安裝，AWS IoT Greengrass 核心軟體才能執行 Python 指令碼。例如，如果您使用圖形 Python 安裝程式，請執行下列動作：

1. 選取安裝所有使用者的啟動器（建議）。
2. 選擇 Customize installation。
3. 選擇 Next。
4. 選取 Install for all users。
5. 選取 Add Python to environment variables。
6. 選擇 Install (安裝)。

如需詳細資訊，請參閱 [Python 3 文件中的在 Windows 上使用 Python](#)。

- AWS Command Line Interface (AWS CLI) 在您的開發電腦和裝置上安裝並設定憑證。請務必使用相同的 AWS 區域，AWS CLI 在開發電腦和裝置上設定。若要 AWS IoT Greengrass V2 搭配使用 AWS CLI，您必須具有下列其中一個版本或更新版本：

- Minimum AWS CLI V1 版本：1.18.197 版
- Minimum AWS CLI V2 版本：v2.1.11

 Tip

您可以執行下列命令來檢查您擁有的 AWS CLI 版本。

```
aws --version
```

如需詳細資訊，請參閱AWS Command Line Interface 《使用者指南》中的[安裝、更新和解除安裝 AWS CLI](#) 和[設定 AWS CLI](#)。

 Note

如果您使用 32 位元 ARM 裝置，例如具有 32 位元作業系統的 Raspberry Pi，則 32 位元 ARM 裝置無法使用 install AWS CLI V1. AWS CLI V2。如需詳細資訊，請參閱[安裝、更新和解除安裝 AWS CLI 版本 1](#)。

步驟 1：設定 AWS 帳戶

註冊 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

AWS 會在註冊程序完成後傳送確認電子郵件給您。您可以隨時登錄 <https://aws.amazon.com/> 並選擇我的帳戶，以檢視您目前的帳戶活動並管理帳戶。

建立具有管理存取權的使用者

註冊後 AWS 帳戶，請保護 AWS 帳戶根使用者、啟用 AWS IAM Identity Center 和建立管理使用者，以免將根使用者用於日常任務。

保護您的 AWS 帳戶根使用者

1. 選擇根使用者並輸入 AWS 帳戶 您的電子郵件地址，以帳戶擁有者 [AWS 管理主控台](#) 身分登入。在下一頁中，輸入您的密碼。

如需使用根使用者登入的說明，請參閱 AWS 登入 使用者指南中的 [以根使用者身分登入](#)。

2. 若要在您的根使用者帳戶上啟用多重要素驗證 (MFA)。

如需說明，請參閱《IAM 使用者指南》中的 [為您的 AWS 帳戶 根使用者（主控台）啟用虛擬 MFA 裝置](#)。

建立具有管理存取權的使用者

1. 啟用 IAM Identity Center。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的 [啟用 AWS IAM Identity Center](#)。

2. 在 IAM Identity Center 中，將管理存取權授予使用者。

如需使用 IAM Identity Center 目錄 做為身分來源的教學課程，請參閱 AWS IAM Identity Center 《使用者指南》中的 [使用預設值設定使用者存取 IAM Identity Center 目錄](#)。

以具有管理存取權的使用者身分登入

- 若要使用您的 IAM Identity Center 使用者簽署，請使用建立 IAM Identity Center 使用者時傳送至您電子郵件地址的簽署 URL。

如需使用 IAM Identity Center 使用者登入的說明，請參閱 AWS 登入 《使用者指南》中的 [登入 AWS 存取入口網站](#)。

指派存取權給其他使用者

1. 在 IAM Identity Center 中，建立一個許可集來遵循套用最低權限的最佳實務。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[建立許可集](#)。

2. 將使用者指派至群組，然後對該群組指派單一登入存取權。

如需指示，請參閱《AWS IAM Identity Center 使用者指南》中的[新增群組](#)。

步驟 2：設定您的環境

Note

這些步驟不適用於 nucleus lite。

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置 (Raspberry Pi)

這些步驟假設您使用 Raspberry Pi 搭配 Raspberry Pi 作業系統。如果您使用不同的裝置或作業系統，請參閱裝置的相關文件。

為 設定 Raspberry Pi AWS IoT Greengrass V2

1. 在 Raspberry Pi 上啟用 SSH 以遠端連線到它。如需詳細資訊，請參閱 Raspberry Pi 文件中的[SSH \(安全殼層\)](#)。
2. 尋找 Raspberry Pi 的 IP 地址，以使用 SSH 與其連線。若要這樣做，您可以在 Raspberry Pi 上執行下列命令。

```
hostname -I
```

3. 使用 SSH 連線至 Raspberry Pi。

在您的開發電腦上，執行下列命令。將#####取代為登入的使用者名稱，並將 *pi-ip-address* 取代為您在上一個步驟中找到的 IP 地址。

```
ssh username@pi-ip-address
```

⚠ Important

如果您的開發電腦使用舊版 Windows，您可能沒有 ssh 命令，或者您可能有，ssh 但無法連線到 Raspberry Pi。若要連線至 Raspberry Pi，您可以安裝和設定 [PuTTY](#)，這是免費的開放原始碼 SSH 用戶端。請參閱 [PuTTY 文件](#) 以連接至 Raspberry Pi。

4. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。在 Raspberry Pi 上，使用下列命令來安裝 Java 11。

```
sudo apt install default-jdk
```

安裝完成後，請執行下列命令來驗證 Java 是否在您的 Raspberry Pi 上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。輸出看起來可能會類似下列範例。

```
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)
```

💡 秘訣：在 Raspberry Pi 上設定核心參數

如果您的裝置是 Raspberry Pi，您可以完成以下步驟來檢視和更新其 Linux 核心參數：

1. 開啟 /boot/cmdline.txt 檔案。此檔案指定要在 Raspberry Pi 開機時套用的 Linux 核心參數。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來開啟 檔案。

```
sudo nano /boot/cmdline.txt
```

2. 確認 /boot/cmdline.txt 檔案包含下列核心參數。systemd.unified_cgroup_hierarchy=0 參數指定 使用 cgroups v1 而非 cgroups v2。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如果/boot/cmdline.txt檔案不包含這些參數，或包含具有不同值的這些參數，請更新檔案以包含這些參數和值。

3. 如果您更新/boot/cmdline.txt檔案，請重新啟動 Raspberry Pi 以套用變更。

```
sudo reboot
```

設定 Linux 裝置（其他）

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. (選用) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者 (通常為 `root`) 具有使用任何使用者和任何群組執行 `sudo` 的許可。

- a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. (選用) 若要[執行容器化 Lambda 函數](#)，您必須啟用 `cgroups` v1，而且必須啟用和掛載記憶體和裝置 `cgroups`。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 `cgroups` 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 [中的需求清單](#)所示。

設定 Windows 裝置

為設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。

2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```
 - g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是[wmic已棄用命令](#)的 Windows 10 或更新版本，請執行下列 PowerShell 命令。


```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
6. 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將##取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 以Accept同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

步驟 3：安裝 AWS IoT Greengrass 核心軟體

Tip

我們建議您試用 [AWS IoT Greengrass AI Agents Context Pack](#)，以快速設定和實驗 AWS IoT Greengrass。代理程式內容套件可讓 AI 代理程式設定 Greengrassnucleus 和 Nucleus Lite、部署元件，以及疑難排解常見問題。

請依照本節中的步驟，將 Raspberry Pi 設定為可用於本機開發 AWS IoT Greengrass 的核心裝置。在本節中，您會下載並執行安裝程式，執行下列動作來設定裝置的 AWS IoT Greengrass Core 軟體：

- 安裝 Greengrass 核元件。核心是強制性元件，也是在裝置上執行 AWS IoT Greengrass Core 軟體的最低需求。如需詳細資訊，請參閱 [Greengrass 核元件](#)。
- 將裝置註冊為 AWS IoT 物件，並下載允許裝置連線的數位憑證 AWS。如需詳細資訊，請參閱[裝置身分驗證和授權 AWS IoT Greengrass](#)。

- 將裝置的 AWS IoT 物件新增至物件群組，即物件群組或機群 AWS IoT。物件群組可讓您管理 Greengrass 核心裝置的機群。將軟體元件部署至裝置時，您可以選擇部署至個別裝置或裝置群組。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[使用 管理裝置 AWS IoT](#)。
- 建立允許 Greengrass 核心裝置與 AWS 服務互動的 IAM 角色。根據預設，此角色可讓您的裝置與互動 AWS IoT，並將日誌傳送至 Amazon CloudWatch Logs。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。
- 安裝 AWS IoT Greengrass 命令列界面 (greengrass-cli)，可用來測試您在核心裝置上開發的自訂元件。如需詳細資訊，請參閱[Greengrass 命令列界面](#)。

安裝 AWS IoT Greengrass Core 軟體（主控台）

1. 登入 [AWS IoT Greengrass 主控台](#)。
2. 在開始使用 Greengrass 下，選擇設定核心裝置。
3. 在步驟 1：註冊 Greengrass 核心裝置，在 Core 裝置名稱中，輸入 Greengrass 核心裝置的 AWS IoT 物件名稱。如果物件不存在，安裝程式會建立它。
4. 在步驟 2：新增至物件群組以套用連續部署，對於物件群組，選擇您要新增核心裝置的 AWS IoT 物件群組。
 - 如果您選取輸入新的群組名稱，然後在物件群組名稱中，輸入要建立的新群組名稱。安裝程式會為您建立新的群組。
 - 如果您選取選取現有群組，請在物件群組名稱中選擇您要使用的現有群組。
 - 如果您選取無群組，則安裝程式不會將核心裝置新增至物件群組。
5. 在步驟 3：安裝 Greengrass Core 軟體下，完成下列步驟。

Nucleus classic

1. 選擇「Nucleus Classic」作為核心裝置的軟體執行時間。
2. 選擇核心裝置的作業系統：Linux 或 Windows。
3. 將 AWS 登入資料提供給裝置，讓安裝程式可以為您的核心裝置佈建 AWS IoT 和 IAM 資源。為了提高安全性，我們建議您取得 IAM 角色的臨時憑證，該角色僅允許佈建所需的最低許可。如需詳細資訊，請參閱[安裝程式佈建資源的最低 IAM 政策](#)。

Note

安裝程式不會儲存或存放您的登入資料。

在您的裝置上，執行下列其中一項作業來擷取憑證，並將其提供給 AWS IoT Greengrass Core 軟體安裝程式：

- （建議）使用來自的臨時登入資料 AWS IAM Identity Center
 - a. 從 IAM Identity Center 提供存取金鑰 ID、私密存取金鑰和工作階段權杖。如需詳細資訊，請參閱《IAM Identity Center 使用者指南》中的[取得和重新整理臨時憑證](#)中的手動憑證重新整理。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass Core 軟體。

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- 使用來自 IAM 角色的臨時安全登入資料：
 - a. 提供您擔任之 IAM 角色的存取金鑰 ID、私密存取金鑰和工作階段權杖。如需如何擷取這些登入資料的詳細資訊，請參閱《IAM 使用者指南》中的[請求臨時安全登入資料](#)。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass Core 軟體。

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- 使用來自 IAM 使用者的長期登入資料：
 - a. 為您的 IAM 使用者提供存取金鑰 ID 和私密存取金鑰。您可以建立 IAM 使用者來佈建您稍後刪除的項目。如需提供使用者的 IAM 政策，請參閱 [安裝程式佈建資源的最低 IAM 政策](#)。如需如何擷取長期憑證的詳細資訊，請參閱《[IAM 使用者指南](#)》中的[管理 IAM 使用者的存取金鑰](#)。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass Core 軟體。

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=wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJa1rXUtnFEMI/K7MDENG/  
bPxRfiCYEXAMPLEKEY"
```

- c. (選用) 如果您建立 IAM 使用者來佈建 Greengrass 裝置，請刪除該使用者。
 - d. (選用) 如果您使用現有 IAM 使用者的存取金鑰 ID 和私密存取金鑰，請更新使用者的金鑰，使其不再有效。如需詳細資訊，請參閱 AWS Identity and Access Management 使用者指南中的[更新存取金鑰](#)。
4. 在執行安裝程式下，完成下列步驟。
- a. 在下載安裝程式下，選擇複製，並在核心裝置上執行複製的命令。此命令會下載最新版本的 AWS IoT Greengrass Core 軟體，並在您的裝置上解壓縮該軟體。
 - b. 在執行安裝程式下，選擇複製，然後在核心裝置上執行複製的命令。此命令會使用您先前指定的 AWS IoT 物件和物件群組名稱，來執行 AWS IoT Greengrass Core 軟體安裝程式，並為核心裝置設定 AWS 資源。

此命令也會執行下列動作：

- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

- 部署 [AWS IoT Greengrass CLI 元件](#)，這是一種命令列工具，可讓您在核心裝置上開發自訂 Greengrass 元件。
- 指定使用 ggc_user 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 ggc_group 系統群組，而安裝程式會為您建立系統使用者和群組。

當您執行此命令時，您應該會看到下列訊息，指出安裝程式已成功執行。

```
Successfully configured Nucleus with provisioned resource details!  
Configured Nucleus to deploy aws.greengrass.Cli component  
Successfully set up Nucleus as a system service
```

 Note

如果您有 Linux 裝置且尚未[系統化](#)，安裝程式不會將軟體設定為系統服務，也不會看到將核設定為系統服務的成功訊息。

Nucleus lite

1. 選擇 Nucleus lite 做為核心裝置的軟體執行時間。
2. 選取您的裝置設定方法，將您的裝置佈建至 Greengrass 核心裝置。

選項 1：使用套件下載設定裝置（約 1MB）

1. 建立 Greengrass 的 AWS IoT 物件和角色。
2. 下載 zip 檔案，其中包含您的裝置需要連線 AWS IoT 的資源 AWS IoT：
 - 使用 憑證授權單位產生的 AWS IoT憑證和私有金鑰。
 - 為您的裝置啟動 Greengrass 安裝的結構描述檔案。
3. 下載將安裝最新 Greengrass Nucleus lite 執行期到 Raspberry Pi 的套件。
4. 將您的裝置佈建為 AWS IoT Greengrass Core 裝置，並將其連接至 AWS IoT：
 - a. 使用 USB 隨身碟、SCP/FTP 或 SD 卡，將 Greengrass 套件和連線套件傳輸至您的裝置。
 - b. 在裝置的 /GreengrassInstaller 目錄中解壓縮 greengrass-package.zip 檔案。
 - c. 在裝置的 /directory 中解壓縮連線套件 zip 檔案。
 - d. 在裝置上執行提供的命令以安裝 AWS IoT Greengrass
5. 然後選擇檢視核心裝置。

選項 2：使用預先設定的全磁碟範例映像下載來設定裝置（約 100MB）

1. 建立 Greengrass 的 AWS IoT 物件和角色。

2. 下載 zip 檔案，其中包含您的裝置需要連線 AWS IoT 的資源 AWS IoT：
 - 使用 憑證授權單位產生的 AWS IoT憑證和私有金鑰。
 - 為您的裝置啟動 Greengrass 安裝的結構描述檔案。
3. 下載包含 Greengrass 和作業系統的預先設定全磁碟範例映像。
 - a. 若要將連線套件傳輸並刷新映像到您的裝置上，請遵循隨映像下載的讀我檔案。
 - b. 若要啟動 Greengrass 安裝，請開啟並從閃爍的映像中開機裝置
4. 然後，選擇檢視核心裝置。

選項 3：使用您自己的自訂建置設定裝置

1. 建立 Greengrass 的 AWS IoT 物件和角色。
2. 下載 zip 檔案，其中包含您的裝置需要連線 AWS IoT 的資源 AWS IoT：
 - 使用 憑證授權單位產生的 AWS IoT憑證和私有金鑰。
 - 為您的裝置啟動 Greengrass 安裝的結構描述檔案。
3. 若要使用來源程式碼中的 Yocto 自訂和建置您自己的映像，然後使用連線套件安裝 nucleus lite，請遵循 GitHub 上的指示。
 - 然後，選擇檢視核心裝置。

安裝 AWS IoT Greengrass 核心軟體 (CLI)

Note

這些步驟不適用於 nucleus lite。

安裝和設定 AWS IoT Greengrass Core 軟體

1. 在您的 Greengrass 核心裝置上，執行下列命令以切換到主目錄。

Linux or Unix

```
cd ~
```

Windows Command Prompt (CMD)

```
cd %USERPROFILE%
```

PowerShell

```
cd ~
```

2. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為 `greengrass-nucleus-latest.zip` 的檔案。

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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. 執行下列命令來啟動 AWS IoT Greengrass Core 軟體安裝程式。此命令會執行下列動作：

- 建立核心裝置操作所需的 AWS 資源。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

- 部署 [AWS IoT Greengrass CLI 元件](#)，這是一種命令列工具，可讓您在核心裝置上開發自訂 Greengrass 元件。
- 指定使用 `ggc_user` 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 `ggc_group` 系統群組，而安裝程式會為您建立系統使用者和群組。

如下所示取代命令中的引數值。

- /greengrass/v2* 或 *C#\greengrass\v2*：用來安裝 AWS IoT Greengrass Core 軟體的根資料夾路徑。
- GreengrassInstaller*。解壓縮 AWS IoT Greengrass Core 軟體安裝程式的資料夾路徑。
- ##*。尋找或建立資源 AWS 區域的。
- MyGreengrassCore*。Greengrass 核心裝置的 AWS IoT 物件名稱。如果物件不存在，安裝程式會建立它。安裝程式會下載憑證以驗證做為 AWS IoT 物件。如需詳細資訊，請參閱 [裝置身分驗證和授權 AWS IoT Greengrass](#)。

Note

物件名稱不能包含冒號 (:) 字元。

- e. ***MyGreengrassCoreGroup***。Greengrass 核心裝置的 AWS IoT 物件群組名稱。如果物件群組不存在，安裝程式會建立該物件並將其新增至該物件。如果物件群組存在且具有作用中部署，核心裝置會下載並執行部署指定的軟體。

Note

物件群組名稱不能包含冒號 (:) 字元。

- f. ***GreengrassV2IoTThingPolicy***。允許 Greengrass 核心裝置與 AWS IoT 和 通訊 AWS IoT 的政策名稱 AWS IoT Greengrass。如果 AWS IoT 政策不存在，安裝程式會使用此名稱建立寬鬆 AWS IoT 政策。您可以限制此政策的使用案例許可。如需詳細資訊，請參閱[AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。
- g. ***GreengrassV2TokenExchangeRole***。允許 Greengrass 核心裝置取得臨時 AWS 憑證的 IAM 角色名稱。如果角色不存在，安裝程式會建立該角色，並建立和連接名為的政策 ***GreengrassV2TokenExchangeRoleAccess***。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。
- h. ***GreengrassCoreTokenExchangeRoleAlias***。IAM 角色的別名，允許 Greengrass 核心裝置稍後取得臨時憑證。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM 角色。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

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

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體量。若要控制記憶體配置，您可以在 核子元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。

當您執行此命令時，您應該會看到下列訊息，指出安裝程式已成功執行。

```
Successfully configured Nucleus with provisioned resource details!  
Configured Nucleus to deploy aws.greengrass.Cli component  
Successfully set up Nucleus as a system service
```

 Note

如果您有 Linux 裝置且尚未[系統化](#)，安裝程式不會將軟體設定為系統服務，也不會看到將核設定為系統服務的成功訊息。

(選用) 執行 Greengrass 軟體 (Linux)

 Note

這些步驟不適用於 nucleus lite。

如果您將軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須執行軟體。若要查看安裝程式是否將軟體設定為系統服務，請在安裝程式輸出中尋找下列行。

```
Successfully set up Nucleus as a system service
```

如果您沒有看到此訊息，請執行下列動作來執行軟體：

1. 執行下列命令來執行軟體。

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

如果成功啟動，軟體會列印下列訊息。

```
Launched Nucleus successfully.
```

2. 您必須讓目前的命令 shell 保持開啟狀態，才能讓 AWS IoT Greengrass Core 軟體保持執行狀態。如果您使用 SSH 連線到核心裝置，請在開發電腦上執行下列命令，以開啟第二個 SSH 工

作階段，供您在核心裝置上執行其他命令。將#####取代為登入的使用者名稱，並將 *pi-ip-address* 取代為裝置的 IP 地址。

```
ssh username@pi-ip-address
```

如需如何與 Greengrass 系統服務互動的詳細資訊，請參閱 [將 Greengrass 核設定為系統服務](#)。

在裝置上驗證 Greengrass CLI 安裝

Note

這些步驟不適用於 nucleus lite。

Greengrass CLI 最多可能需要一分鐘的時間才能部署。執行下列命令來檢查部署的狀態。以核心裝置的名稱取代 *MyGreengrassCore*。

```
aws greengrassv2 list-effective-deployments --core-device-thing-name MyGreengrassCore
```

coreDeviceExecutionStatus 指出核心裝置的部署狀態。當狀態為 SUCCEEDED，請執行下列命令，以確認 Greengrass CLI 已安裝並執行。*/greengrass/v2* 將取代為根資料夾的路徑。

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
```

命令輸出 Greengrass CLI 的說明資訊。如果 greengrass-cli 找不到，則部署可能已無法安裝 Greengrass CLI。如需詳細資訊，請參閱 [故障診斷 AWS IoT Greengrass V2](#)。

您也可以執行下列命令，將 AWS IoT Greengrass CLI 手動部署至您的裝置。

- 將 **##** 取代 AWS 區域 為您使用的。請確定您使用 AWS 區域 與 AWS CLI 在裝置上設定 相同的。
- 將 **account-id** 取代為您的 AWS 帳戶 ID。
- 以核心裝置的名稱取代 **MyGreengrassCore**。

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

您可以將 **/greengrass/v2/bin**(Linux) 或 **C:\greengrass\v2\bin**(Windows) 新增至 PATH 環境變數，以便在沒有絕對路徑 greengrass-cli 的情況下執行。

AWS IoT Greengrass 核心軟體和本機開發工具會在您的裝置上執行。接下來，您可以在裝置上開發 Hello World AWS IoT Greengrass 元件。

步驟 4：在您的裝置上開發和測試元件

元件是在 AWS IoT Greengrass 核心裝置上執行的軟體模組。元件可讓您將複雜的應用程式建立和管理為分散的建置區塊，以便從一個 Greengrass 核心裝置重複使用。每個元件都由配方和成品組成。

- 配方

每個元件都包含配方檔案，定義其中繼資料。配方也會指定元件的組態參數、元件相依性、生命週期和平台相容性。元件生命週期會定義安裝、執行和關閉元件的命令。如需詳細資訊，請參閱[AWS IoT Greengrass 元件配方參考](#)。

您可以 [JSON](#) 或 [YAML](#) 格式定義配方。

- 成品

元件可以有任意數量的成品，這些成品是元件二進位檔。成品可以包含指令碼、編譯的程式碼、靜態資源，以及元件使用的任何其他檔案。元件也可以使用元件相依性的成品。

透過 AWS IoT Greengrass，您可以使用 Greengrass CLI 在 Greengrass 核心裝置上於本機開發和測試元件，而無需與 AWS 雲端互動。當您完成本機元件時，您可以使用元件配方和成品在 AWS 雲端的 AWS IoT Greengrass 服務中建立該元件，然後將其部署到所有 Greengrass 核心裝置。如需元件的詳細資訊，請參閱 [開發 AWS IoT Greengrass 元件](#)。

在本節中，您將了解如何在核心裝置上於本機建立和執行基本 Hello World 元件。

在裝置上開發 Hello World 元件

1. 使用配方和成品的子資料夾為您的元件建立資料夾。在 Greengrass 核心裝置上執行下列命令，以建立這些資料夾並變更為元件資料夾。將 `~/greengrassv2` 或 `%USERPROFILE%\greengrassv2` 取代為用於本機開發的資料夾路徑。

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. 使用文字編輯器來建立配方檔案，定義元件的中繼資料、參數、相依性、生命週期和平台功能。在配方檔案名稱中包含元件版本，讓您可以識別哪個配方會反映哪個元件版本。您可以為您的配方選擇 YAML 或 JSON 格式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

JSON

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

YAML

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

Note

AWS IoT Greengrass 使用元件的語意版本。語意版本遵循 major.minor.patch 號碼系統。例如，版本1.0.0代表元件的第一個主要版本。如需詳細資訊，請參閱[語意版本規格](#)。

3. 將下列配方貼到 檔案中。

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}"
```

此配方的 `ComponentConfiguration` 區段定義 參數 `Message`，預設為 `world`。`Manifests` 區段定義資訊清單，這是一組平台的生命週期指示和成品。您可以定義多個資訊清單，以指定各種平台的不同安裝指示，例如。在資訊清單中，`Lifecycle`區段會指示 Greengrass 核心裝置以`Message`參數值做為引數來執行 `Hello World` 指令碼。

4. 執行下列命令來建立元件成品的資料夾。

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

您必須針對成品資料夾路徑使用下列格式。包含您在配方中指定的元件名稱和版本。

```
artifacts/componentName/componentVersion/
```

5. 使用文字編輯器為您的 `Hello World` 元件建立 Python 指令碼成品檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

將下列 Python 指令碼複製並貼到 檔案中。

```
import sys
```

```
message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

6. 使用本機 AWS IoT Greengrass CLI 來管理 Greengrass 核心裝置上的元件。

執行下列命令，將元件部署至 AWS IoT Greengrass 核心。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為您的 AWS IoT Greengrass V2 根資料夾，並將 `~/greengrassv2` 或 `%USERPROFILE%\greengrassv2` 取代為您的元件開發資料夾。

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"
```

此命令會新增在 中使用配方的元件，recipes 以及在 中使用 Python 指令碼的元件 artifacts。--merge 選項會新增或更新您指定的元件和版本。

7. Core AWS IoT Greengrass 軟體會將 stdout 從元件程序儲存至 logs 資料夾中的日誌檔案。執行下列命令來驗證 Hello World 元件是否執行並列印訊息。

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
```

type 命令會將檔案的內容寫入終端機。多次執行此命令，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, world!
```

Note

如果檔案不存在，則本機部署可能尚未完成。如果檔案在 15 秒內不存在，則部署可能會失敗。例如，如果您的配方無效，就可能會發生這種情況。執行下列命令來檢視 AWS IoT Greengrass 核心日誌檔案。此檔案包含 Greengrass 核心裝置部署服務的日誌。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.log
```

type 命令會將檔案的內容寫入終端機。多次執行此命令，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

8. 修改本機元件以反覆執行和測試程式碼。在文字編輯器 `hello_world.py` 中開啟，並在第 4 行新增下列程式碼，以編輯 AWS IoT Greengrass 核心記錄的訊息。

```
message += " Greetings from your first Greengrass component."
```

`hello_world.py` 指令碼現在應具有下列內容。

```
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. 執行下列命令，以您的變更更新元件。

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"
```

此命令會使用最新的 Hello World 成品更新 `com.example.HelloWorld` 元件。

10. 執行下列命令以重新啟動元件。當您重新啟動元件時，核心裝置會使用最新的變更。

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. 再次檢查日誌，確認 Hello World 元件列印新訊息。

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
```

`type` 命令會將檔案的內容寫入終端機。多次執行此命令，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, world! Greetings from your first Greengrass component.
```

12. 您可以更新元件的組態參數，以測試不同的組態。部署元件時，您可以指定組態更新，以定義如何修改核心裝置上的元件組態。您可以指定要重設為預設值的組態值，以及要合併至核心裝置的新組態值。如需詳細資訊，請參閱[更新元件組態](#)。

請執行下列操作：

- a. 使用文字編輯器來建立名為 `hello-world-config-update.json` 的檔案，以包含組態更新

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano hello-world-config-update.json
```

- b. 將下列 JSON 物件複製並貼入 檔案。此 JSON 物件會定義組態更新，將值合併至 `Message` 參數以更新其值。此組態更新不會指定任何要重設的值。您不需要重設 `Message` 參數，因為合併更新會取代現有的值。

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "Message": "friend"
    }
  }
}
```

- c. 執行下列命令，將組態更新部署到 Hello World 元件。

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. 再次檢查日誌，確認 Hello World 元件輸出新訊息。

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
```

type 命令會將檔案的內容寫入終端機。多次執行此命令，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

您應該會看到類似下列範例的訊息。

```
Hello, friend! Greetings from your first Greengrass component.
```

13. 完成元件測試後，請將其從核心裝置中移除。執行下列命令。

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

上傳元件到之後，您需要此步驟，才能將元件部署回核心裝置 AWS IoT Greengrass。否則，部署會失敗並出現版本相容性錯誤，因為本機部署會指定不同版本的元件。

執行下列命令，並確認com.example.HelloWorld元件未出現在裝置上的元件清單中。

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
```

您的 Hello World 元件已完成，您現在可以將其上傳至 AWS IoT Greengrass 雲端服務。然後，您可以將元件部署到 Greengrass 核心裝置。

步驟 5：在 AWS IoT Greengrass 服務中建立元件

當您在核心裝置上完成元件的開發時，您可以將其 AWS IoT Greengrass 上傳至 中的 服務 AWS 雲端。您也可以直接在 [AWS IoT Greengrass 主控台](#) 中建立元件。AWS IoT Greengrass 提供元件管理服務，可託管您的元件，以便將元件部署到個別裝置或裝置機群。若要將元件上傳至 AWS IoT Greengrass 服務，請完成下列步驟：

- 將元件成品上傳至 S3 儲存貯體。

- 將每個成品的 Amazon Simple Storage Service (Amazon S3) URI 新增至元件配方。
- AWS IoT Greengrass 從元件配方在 中建立元件。

在本節中，您可以在 Greengrass 核心裝置上完成這些步驟，將 Hello World 元件上傳至 AWS IoT Greengrass 服務。

在 AWS IoT Greengrass（主控台）中建立元件

1. 使用您 AWS 帳戶中的 S3 儲存貯體來託管 AWS IoT Greengrass 元件成品。當您將元件部署到核心裝置時，裝置會從儲存貯體下載元件的成品。

您可以使用現有的 S3 儲存貯體，也可以建立新的儲存貯體。

- a. 在 [Amazon S3 主控台](#) 的儲存貯體下，選擇建立儲存貯體。
- b. 針對儲存貯體名稱，輸入唯一的儲存貯體名稱。例如，您可以使用 **greengrass-component-artifacts-*region*-123456789012**。將 **123456789012** 取代 AWS 區域 為您在本教學課程中使用的 AWS 帳戶 ID 和 ##。
- c. 針對 AWS 區域，選取您用於本教學課程 AWS 的區域。
- d. 選擇建立儲存貯體。
- e. 在儲存貯體下，選擇您建立的儲存貯體，將 `hello_world.py` 指令碼上傳至儲存貯體中的 `artifacts/com.example.HelloWorld/1.0.0` 資料夾。如需將物件上傳至 S3 儲存貯體的資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的 [上傳物件](#)。
- f. 複製 S3 儲存貯體中 `hello_world.py` 物件的 S3 URI。此 URI 看起來應該類似下列範例。將 `amzn-s3-demo-bucket` 取代為 S3 儲存貯體的名稱。

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

2. 允許核心裝置存取 S3 儲存貯體中的元件成品。

每個核心裝置都有一個 [核心裝置 IAM 角色](#)，允許它與 互動 AWS IoT 並將日誌傳送到 AWS 雲端。根據預設，此裝置角色不允許存取 S3 儲存貯體，因此您必須建立並連接允許核心裝置從 S3 儲存貯體擷取元件成品的政策。

如果您裝置的角色已允許存取 S3 儲存貯體，您可以略過此步驟。否則，請建立允許存取的 IAM 政策，並將其連接到角色，如下所示：

- a. 在 [IAM 主控台](#) 導覽功能表中，選擇政策，然後選擇建立政策。

- b. 在 JSON 標籤上，用以下政策取代預留位置內容。將 `amzn-s3-demo-bucket` 取代為 S3 儲存貯體的名稱，其中包含要下載的核心裝置元件成品。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

- c. 選擇下一步。
 - d. 在政策詳細資訊區段中，針對名稱輸入 **MyGreengrassV2ComponentArtifactPolicy**。
 - e. 選擇建立政策。
 - f. 在 [IAM 主控台](#) 導覽功能表中，選擇角色，然後選擇核心裝置的角色名稱。您在安裝 AWS IoT Greengrass 核心軟體時指定了此角色名稱。如果您未指定名稱，則預設值為 `GreengrassV2TokenExchangeRole`。
 - g. 在許可下，選擇新增許可，然後選擇連接政策。
 - h. 在新增許可頁面上，選取您建立 `MyGreengrassV2ComponentArtifactPolicy` 的政策旁的核取方塊，然後選擇新增許可。
3. 使用元件配方在 [AWS IoT Greengrass 主控台](#) 中建立元件。
 - a. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇元件，然後選擇建立元件。
 - b. 在元件資訊下，選擇將配方輸入為 JSON。預留位置配方看起來應該類似下列範例。

```
{
  "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. 將每個Artifacts區段中的預留位置 URI 取代為hello_world.py物件的 S3 URI。
- d. 選擇建立元件。
- e. 在 com.example.HelloWorld 元件頁面上，確認元件的狀態為可部署。

在 AWS IoT Greengrass (AWS CLI) 中建立元件

上傳您的 Hello World 元件

1. 在 中使用 S3 儲存貯體 AWS 帳戶 來託管 AWS IoT Greengrass 元件成品。當您將元件部署到核心裝置時，裝置會從儲存貯體下載元件的成品。

您可以使用現有的 S3 儲存貯體，或執行下列命令來建立儲存貯體。此命令會使用您的 AWS 帳戶 ID 和 建立儲存貯體 AWS 區域，以形成唯一的儲存貯體名稱。將 **123456789012** 取代為您在本教學課程中使用的 AWS 帳戶 ID AWS 區域 和 **##**。

```
aws s3 mb s3://greengrass-component-artifacts-123456789012-region
```

如果請求成功，命令會輸出下列資訊。

```
make_bucket: greengrass-component-artifacts-123456789012-region
```

2. 允許核心裝置存取 S3 儲存貯體中的元件成品。

每個核心裝置都有一個[核心裝置 IAM 角色](#)，允許它與 互動 AWS IoT 並將日誌傳送到 AWS 雲端。根據預設，此裝置角色不允許存取 S3 儲存貯體，因此您必須建立並連接允許核心裝置從 S3 儲存貯體擷取元件成品的政策。

如果核心裝置的角色已允許存取 S3 儲存貯體，您可以略過此步驟。否則，請建立允許存取的 IAM 政策，並將其連接到角色，如下所示：

- a. 建立名為 的檔案 `component-artifact-policy.json`，並將下列 JSON 複製到 檔案。此政策允許存取 S3 儲存貯體中的所有檔案。將 `amzn-s3-demo-bucket` 取代為 S3 儲存貯體的名稱。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

```
}
```

- b. 執行下列命令，從 中的政策文件建立政策 `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
```

從輸出中的政策中繼資料複製政策 Amazon Resource Name (ARN)。您可以在下一個步驟中使用此 ARN 將此政策連接至核心裝置角色。

- c. 執行下列命令，將政策連接至核心裝置角色。以核心裝置的角色名稱取代 *GreengrassV2TokenExchangeRole*。您在安裝 AWS IoT Greengrass 核心軟體時指定了此角色名稱。將政策 ARN 取代為上一個步驟的 ARN。

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
```

如果命令沒有輸出，表示成功。核心裝置現在可以存取您上傳至此 S3 儲存貯體的成品。

3. 將 Hello World Python 指令碼成品上傳至 S3 儲存貯體。

執行下列命令，將指令碼上傳至您 AWS IoT Greengrass 核心上存在指令碼的 儲存貯體中的相同路徑。將 `amzn-s3-demo-bucket` 取代為 S3 儲存貯體的名稱。

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
```

如果請求成功，命令 `upload:` 會輸出以 `開頭` 的行。

4. 將成品的 Amazon S3 URI 新增至元件配方。

Amazon S3 URI 由儲存貯體名稱和儲存貯體中成品物件的路徑組成。指令碼成品的 Amazon S3 URI 是您在上一個步驟中上傳成品的 URI。此 URI 看起來應該類似下列範例。將 `amzn-s3-demo-bucket` 取代為 S3 儲存貯體的名稱。

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

若要將成品新增至配方，請新增包含具有 Artifacts Amazon S3 URI 結構的清單。

JSON

```
"Artifacts": [  
  {  
    "URI": "s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/  
hello_world.py"  
  }  
]
```

在文字編輯器中開啟配方檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

將成品新增至配方。您的配方檔案看起來應該類似下列範例。

```
{  
  "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

```

在文字編輯器中開啟配方檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

將成品新增至配方。您的配方檔案看起來應該類似下列範例。

```
---
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. AWS IoT Greengrass 從配方在 中建立元件資源。執行下列命令，從您提供的二進位檔案配方建立 元件。

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
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "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": {}
  }
}
```

arn 從輸出複製，以在下一個步驟中檢查元件的狀態。

Note

您也可以在此 [AWS IoT Greengrass 主控台](#) 的元件頁面上看到 Hello World 元件。

6. 確認元件已建立並準備好部署。當您建立元件時，其狀態為 REQUESTED。然後，AWS IoT Greengrass 驗證元件可部署。您可以執行下列命令來查詢元件狀態，並確認元件可部署。將取代arn為上一個步驟的 ARN。

```
aws greengrassv2 describe-component --arn
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0"
```

如果元件驗證，回應會指出元件狀態為 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"
    }
  ]
}
```

您的 Hello World 元件現在可在 中使用 AWS IoT Greengrass。您可以將其部署回此 Greengrass 核心裝置或其他核心裝置。

步驟 6：部署您的元件

使用 AWS IoT Greengrass，您可以將元件部署到個別裝置或裝置群組。當您部署元件時，會在每個目標裝置上 AWS IoT Greengrass 安裝並執行該元件的軟體。您可以指定要部署的元件，以及要為每個元件部署的組態更新。您也可以控制部署如何推展到部署目標的裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

在本節中，您將 Hello World 元件部署回 Greengrass 核心裝置。

部署您的元件（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
2. 在元件頁面上的我的元件索引標籤上，選擇 com.example.HelloWorld。
3. 在 com.example.HelloWorld 頁面中，選擇部署。
4. 從新增至部署，選擇建立新部署，然後選擇下一步。
5. 在指定目標頁面上，執行下列作業：
 - a. 在 Name (名稱) 方塊中，輸入 **Deployment for MyGreengrassCore**。
 - b. 針對部署目標，選擇核心裝置，以及核心裝置的 AWS IoT 物件名稱。本教學課程中的預設值為 *MyGreengrassCore*。
 - c. 選擇 Next (下一步)。
6. 在選取元件頁面的我的元件下，確認已選取 com.example.HelloWorld 元件，然後選擇下一步。

7. 在設定元件頁面上，選擇 `com.example.HelloWorld`，然後執行下列動作：
 - a. 選擇設定元件。
 - b. 在組態更新下的要合併的組態中，輸入下列組態。

```
{  
  "Message": "universe"  
}
```

此組態更新會將 Hello World Message 參數 `universe` 設定為此部署中裝置的。

- c. 選擇確認。
 - d. 選擇 Next (下一步)。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。
10. 確認部署已成功完成。可能需要幾分鐘才能完成部署。檢查 Hello World 日誌以驗證變更。在 Greengrass 核心裝置上執行下列命令。

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
```

您應該會看到類似下列範例的訊息。

```
Hello, universe! Greetings from your first Greengrass component.
```

Note

如果日誌訊息未變更，則部署會失敗或無法連線到核心裝置。如果您的核心裝置未連線至網際網路，或沒有從 S3 儲存貯體擷取成品的許可，就可能發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass 核心軟體日誌檔案。此檔案包含 Greengrass 核心裝置的部署服務日誌。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.log
```

type 命令會將檔案的內容寫入終端機。執行此命令多次，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

如需詳細資訊，請參閱[故障診斷 AWS IoT Greengrass V2](#)。

部署您的元件 (AWS CLI)

部署 Hello World 元件

1. 在您的開發電腦上，建立名為 的檔案，hello-world-deployment.json並將下列 JSON 複製到 檔案。此檔案定義要部署的元件和組態。

```
{
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"Message\":\"universe\"}"
      }
    }
  }
}
```

```
    }
  }
}
```

此組態檔案會指定 部署您在先前程序中開發和發佈1.0.0的 Hello World 元件版本。configurationUpdate 指定在 JSON 編碼字串中合併元件組態。此組態更新會將 Hello World Message 參數設定為此部署中裝置的 universe 。

2. 執行下列命令，將元件部署到您的 Greengrass 核心裝置。您可以部署到物件，也就是個別裝置，或是物件群組，也就是裝置群組。將 *MyGreengrassCore* 取代為核心裝置的 AWS IoT 物件名稱。

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
```

命令會輸出類似下列範例的回應。

```
{
  "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. 確認部署已成功完成。可能需要幾分鐘才能完成部署。檢查 Hello World 日誌以驗證變更。在 Greengrass 核心裝置上執行下列命令。

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
```

您應該會看到類似下列範例的訊息。

```
Hello, universe! Greetings from your first Greengrass component.
```

Note

如果日誌訊息未變更，則部署會失敗或無法連線到核心裝置。如果您的核心裝置未連線至網際網路，或沒有從 S3 儲存貯體擷取成品的許可，就可能發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass 核心軟體日誌檔案。此檔案包含 Greengrass 核心裝置的部署服務日誌。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\greengrass.log
```

type 命令會將檔案的內容寫入終端機。執行此命令多次，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
```

如需詳細資訊，請參閱[故障診斷 AWS IoT Greengrass V2](#)。

後續步驟

您已完成本教學課程。AWS IoT Greengrass 核心軟體和您的 Hello World 元件會在您的裝置上執行。此外，您的 Hello World 元件可在 AWS IoT Greengrass 雲端服務中部署到其他裝置。如需本教學課程探索主題的詳細資訊，請參閱以下內容：

- [建立 AWS IoT Greengrass 元件](#)
- [發佈元件以部署到您的核心裝置](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)

設定 AWS IoT Greengrass 核心裝置

完成本節中的任務，以安裝、設定和執行 AWS IoT Greengrass 核心軟體。

Note

本節說明 AWS IoT Greengrass Core 軟體的進階安裝和組態。這些步驟不適用於 nucleus lite。如果您是第一次使用 AWS IoT Greengrass V2，我們建議您先完成[入門教學](#)課程，以設定核心裝置並探索 的功能 AWS IoT Greengrass。

主題

- [支援平台](#)
- [裝置要求](#)
- [Lambda 函數需求](#)
- [設定 AWS 帳戶](#)
- [安裝 AWS IoT Greengrass Core 軟體](#)
- [執行 AWS IoT Greengrass 核心軟體](#)
- [在 Docker 容器中執行 AWS IoT Greengrass 核心軟體](#)
- [設定 AWS IoT Greengrass 核心軟體](#)
- [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)
- [解除安裝 AWS IoT Greengrass Core 軟體](#)

支援平台

- [Greengrass 核支援的平台](#)
- [Greengrass nucleus lite 支援的平台](#)

裝置要求

- [Greengrass 核設備需求](#)
- [Greengrass nucleus lite 裝置需求](#)

Lambda 函數需求

Important

Greengrass Lambda 函數目前不支援 Greengrass nucleus lite。

您的裝置必須符合下列要求，才能執行 Lambda 函數：

- 以 Linux 為基礎的作業系統。
- 您的裝置必須具有 `mkfifo shell` 命令。
- 您的裝置必須執行 Lambda 函數所需的程式設計語言程式庫。您必須在裝置上安裝所需的程式庫，並將其新增至 `PATH` 環境變數。Greengrass 支援所有 Lambda 支援的 Python、Node.js 和 Java 執行時間版本。Greengrass 不會對已取代的 Lambda 執行時間版本套用任何其他限制。如需 Lambda 執行時間 AWS IoT Greengrass 支援的詳細資訊，請參閱 [執行 AWS Lambda 函數](#)。
- 若要執行容器化 Lambda 函數，您的裝置必須符合下列要求：
 - Linux 核心版本 4.4 或更新版本。
 - 核心必須支援 [cgroups](#) v1，而且您必須啟用並掛載下列 cgroups：
 - 的記憶體 cgroup AWS IoT Greengrass，用於設定容器化 Lambda 函數的記憶體限制。
 - 容器化 Lambda 函數的裝置 cgroup 可存取系統裝置或磁碟區。

AWS IoT Greengrass 核心軟體不支援 cgroups v2。

若要符合此需求，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

Tip

在 Raspberry Pi 上，編輯 `/boot/cmdline.txt` 檔案以設定裝置的核心參數。

- 您必須在裝置上啟用下列 Linux 核心組態：
 - 命名空間：
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`

- CONFIG_USER_NS
- CONFIG_PID_NS
- Cgroups :
 - CONFIG_CGROUP_DEVICE
 - CONFIG_CGROUPS
 - CONFIG_MEMCG
- 其他 :
 - CONFIG_POSIX_MQUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM

 Tip

檢查 Linux 發行版本的文件，以了解如何驗證和設定 Linux 核心參數。您也可以使用 AWS IoT Device Tester 的 AWS IoT Greengrass 來驗證您的裝置是否符合這些要求。如需詳細資訊，請參閱[使用 AWS IoT Device Tester for AWS IoT Greengrass V2](#)。

設定 AWS 帳戶

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

若要建立管理員使用者，請選擇下列其中一個選項。

選擇一種管理管理員的方式	到	根據	您也可以
在 IAM Identity Center (建議)	使用短期憑證存取 AWS。 這與安全性最佳實務一致。有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務 。	請遵循 AWS IAM Identity Center 使用者指南的 入門 中的說明。	透過在 AWS Command Line Interface 使用者指南中設定 AWS CLI 以使用來設定 AWS IAM Identity Center 程式設計存取。
在 IAM 中 (不建議使用)	使用長期憑證存取 AWS。	請遵循《IAM 使用者指南》中 建立 IAM 使用者以進行緊急存取 的指示。	請依照《IAM 使用者指南》中的 管理 IAM 使用者的存取金鑰 設定以程式設計方式存取。

安裝 AWS IoT Greengrass Core 軟體

AWS IoT Greengrass 延伸 AWS 到邊緣裝置，讓他們可以對產生的資料採取行動，同時使用 AWS 雲端進行管理、分析和耐用儲存。在邊緣裝置上安裝 AWS IoT Greengrass 核心軟體，以與 AWS IoT Greengrass 和整合 AWS 雲端。

Important

在您下載並安裝 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 2.0 版的[要求](#)。

Core AWS IoT Greengrass 軟體包含安裝程式，可將您的裝置設定為 Greengrass 核心裝置。當您執行安裝程式時，您可以設定選項，例如根資料夾和 AWS 區域 要使用的。您可以選擇讓安裝程式為您建立必要的 AWS IoT 和 IAM 資源。您也可以選擇部署本機開發工具，以設定您用於自訂元件開發的裝置。

AWS IoT Greengrass 核心軟體需要下列 AWS IoT 和 IAM 資源才能連線至 AWS 雲端 並操作：

- **實 AWS IoT 物。**當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。此憑證可讓裝置與 AWS IoT 和 通訊 AWS IoT Greengrass。如需詳細資訊，請參閱[的裝置身分驗證和授權 AWS IoT Greengrass](#)。
- **(選用) AWS IoT 物件群組。**您可以使用物件群組來管理 Greengrass 核心裝置的機群。當您將軟體元件部署到裝置時，您可以選擇部署到個別裝置或裝置群組。您可以將裝置新增至物件群組，將該物件群組的軟體元件部署到裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。
- **IAM 角色。**Greengrass 核心裝置使用 AWS IoT Core 登入資料提供者來授權呼叫具有 IAM 角色 AWS 的服務。此角色可讓您的裝置與 互動 AWS IoT、將日誌傳送至 Amazon CloudWatch Logs，以及從 Amazon Simple Storage Service (Amazon S3) 下載自訂元件成品。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。
- **AWS IoT 角色別名。**Greengrass 核心裝置使用角色別名來識別要使用的 IAM 角色。角色別名可讓您變更 IAM 角色，但保持裝置組態相同。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權直接呼叫 AWS 服務](#)。

選擇下列其中一個選項，在您的核心裝置上安裝 AWS IoT Greengrass Core 軟體。

- **快速安裝**

選擇此選項，以盡可能少的步驟設定 Greengrass 核心裝置。安裝程式會為您建立所需的 AWS IoT 和 IAM 資源。此選項需要您提供 AWS 登入資料給安裝程式，以在 中建立資源 AWS 帳戶。

您無法使用此選項在防火牆或網路代理後方安裝。如果您的裝置位於防火牆或網路代理之後，請考慮[手動安裝](#)。

如需詳細資訊，請參閱[使用自動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

- **手動安裝**

選擇此選項可手動建立所需的 AWS 資源，或安裝在防火牆或網路代理後方。透過使用手動安裝，您不需要授予安裝程式在 中建立資源的許可 AWS 帳戶，因為您建立必要的 AWS IoT 和 IAM 資源。您也可以將裝置設定為在連接埠 443 上或透過網路代理進行連線。您也可以將 AWS IoT Greengrass

Core 軟體設定為使用存放在硬體安全模組 (HSM)、信任平台模組 (TPM) 或其他密碼編譯元素中的私有金鑰和憑證。

如需詳細資訊，請參閱[使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

- 使用 AWS IoT 機群佈建進行安裝

選擇此選項，從 AWS IoT 機群佈建範本建立所需的 AWS 資源。您可以選擇此選項在機群中建立類似的裝置，或者如果您製造客戶稍後啟用的裝置，例如車輛或智慧型家用裝置。裝置使用宣告憑證來驗證和佈建 AWS 資源，包括裝置用來連線至以進行正常操作 AWS 雲端的 X.509 用戶端憑證。您可以在製造期間將宣告憑證嵌入或刷新到裝置的硬體，而且您可以使用相同的宣告憑證和金鑰來佈建多個裝置。您也可以設定裝置在連接埠 443 上或透過網路代理進行連線。

如需詳細資訊，請參閱[使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體](#)。

- 使用自訂佈建安裝

選擇此選項可開發自訂 Java 應用程式，以佈建所需的 AWS 資源。如果您[建立自己的 X.509 用戶端憑證](#)，或想要對佈建程序有更多控制權，您可以選擇此選項。AWS IoT Greengrass 提供可實作的界面，以便在自訂佈建應用程式與 AWS IoT Greengrass Core 軟體安裝程式之間交換資訊。

如需詳細資訊，請參閱[使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

AWS IoT Greengrass 也提供執行 AWS IoT Greengrass Core 軟體的容器化環境。您可以使用 Dockerfile 在 [Docker AWS IoT Greengrass 容器中執行](#)。

主題

- [使用自動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)
- [使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)
- [使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體](#)
- [使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體](#)
- [安裝程式引數](#)

使用自動資源佈建安裝 AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體包含安裝程式，可將您的裝置設定為 Greengrass 核心裝置。若要快速設定裝置，安裝程式可以佈建核心裝置操作所需的 AWS IoT 物件、AWS IoT 物件群組、IAM 角色和 AWS IoT 角色別名。安裝程式也可以將本機開發工具部署到核心裝置，因此您可以使用裝置來開發和測試自訂軟體元件。安裝程式需要 AWS 登入資料才能佈建這些資源並建立部署。

如果您無法提供 AWS 登入資料給裝置，您可以佈建核心裝置操作所需的 AWS 資源。您也可以將開發工具部署到核心裝置，以用作開發裝置。這可讓您在執行安裝程式時為裝置提供較少的許可。如需詳細資訊，請參閱[使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [設定裝置環境](#)
- [提供 AWS 登入資料給裝置](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：


```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. （選用）建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者（通常為 root）具有使用任何使用者和任何群組執行 `sudo` 的許可。

- a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. （選用）若要[執行容器化 Lambda 函數](#)，您必須啟用 [cggroups](#) v1，而且必須啟用和掛載記憶體和裝置 `cggroups`。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 中的需求清單所示[裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.0 和更新版本。

為 設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。

- 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是 [wmic 已棄用命令](#) 的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

- 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
- 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將##取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 以Accept同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

提供 AWS 登入資料給裝置

將 AWS 登入資料提供給您的裝置，讓安裝程式可以佈建所需的 AWS 資源。如需所需許可的詳細資訊，請參閱[安裝程式佈建資源的最低 IAM 政策](#)。

提供 AWS 登入資料給裝置

- 將 AWS 登入資料提供給裝置，讓安裝程式可以為您的核心裝置佈建 AWS IoT 和 IAM 資源。為了提高安全性，我們建議您取得 IAM 角色的臨時登入資料，只允許佈建所需的最低許可。如需詳細資訊，請參閱[安裝程式佈建資源的最低 IAM 政策](#)。

Note

安裝程式不會儲存或存放您的登入資料。

在您的裝置上，執行下列其中一項操作來擷取登入資料，並將其提供給 AWS IoT Greengrass Core 軟體安裝程式：

- (建議) 使用來自的臨時登入資料 AWS IAM Identity Center
 - 從 IAM Identity Center 提供存取金鑰 ID、私密存取金鑰和工作階段字符。如需詳細資訊，請參閱《IAM Identity Center 使用者指南》中的[取得和重新整理暫時憑證](#)中的手動憑證重新整理。
 - 執行下列命令，將登入資料提供給 AWS IoT Greengrass 核心軟體。

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- 使用來自 IAM 角色的臨時安全登入資料：
 - a. 提供您擔任之 IAM 角色的存取金鑰 ID、私密存取金鑰和工作階段字符。如需如何擷取這些登入資料的詳細資訊，請參閱《IAM 使用者指南》中的[請求臨時安全登入](#)資料。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass 核心軟體。

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- 使用來自 IAM 使用者的長期登入資料：
 - a. 為您的 IAM 使用者提供存取金鑰 ID 和私密存取金鑰。您可以建立 IAM 使用者來佈建您稍後刪除的項目。如需提供使用者的 IAM 政策，請參閱 [安裝程式佈建資源的最低 IAM 政策](#)。如需如何擷取長期憑證的詳細資訊，請參閱《[IAM 使用者指南](#)》中的[管理 IAM 使用者的存取金鑰](#)。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass 核心軟體。

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. (選用) 如果您建立了 IAM 使用者來佈建 Greengrass 裝置，請刪除該使用者。
- d. (選用) 如果您使用現有 IAM 使用者的存取金鑰 ID 和私密存取金鑰，請更新使用者的金鑰，使其不再有效。如需詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的[更新存取金鑰](#)。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass 核心軟體。將##取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為 `greengrass-nucleus-latest.zip` 的檔案。

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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

2. (選用) 驗證 Greengrass 核軟體簽章

Note

此功能適用於 Greengrass 核版本 2.9.5 和更新版本。

- a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。`jdk17.0.6_10` 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 調用會產生輸出，指出驗證的結果。
 - i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：

```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。
3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

 Important

如果您安裝的 Greengrass 核版本早於 2.4.0 版，請勿在安裝 AWS IoT Greengrass 核心軟體之後移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass Core 軟體後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定 執行下列動作的引數執行安裝程式：

- 建立核心裝置操作所需的 AWS 資源。
- 指定 以使用ggc_user系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用ggc_group系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

 Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

若要使用本機開發工具設定開發裝置，請指定 `--deploy-dev-tools true` 引數。安裝完成後，本機開發工具最多可能需要一分鐘的時間才能部署。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。

安裝 AWS IoT Greengrass 核心軟體

1. 執行 AWS IoT Greengrass Core 安裝程式。取代命令中的引數值，如下所示。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 `C:\greengrass\v2` 或 `D:\greengrass\v2`，將 Greengrass 元件路徑保持在 260 個字元的限制以下。

- a. `/greengrass/v2` 或 `C#\greengrass\v2`：用來安裝 AWS IoT Greengrass Core 軟體的根資料夾路徑。
- b. `GreengrassInstaller`。解壓縮 AWS IoT Greengrass Core 軟體安裝程式的資料夾路徑。
- c. `##`。尋找或建立資源 AWS 區域的。
- d. `MyGreengrassCore`。Greengrass 核心裝置的 AWS IoT 物件名稱。如果物件不存在，安裝程式會建立它。安裝程式會下載憑證以驗證為 AWS IoT 實物。如需詳細資訊，請參閱[裝置身分驗證和授權 AWS IoT Greengrass](#)。

Note

物件名稱不能包含冒號 (:) 字元。

- e. `MyGreengrassCoreGroup`。Greengrass 核心裝置的 AWS IoT 物件群組名稱。如果物件群組不存在，安裝程式會建立它，並將該物件新增至其中。如果物件群組存在且具有作用中部署，核心裝置會下載並執行部署指定的軟體。

Note

物件群組名稱不能包含冒號 (:) 字元。

- f. *GreengrassV2IoTThingPolicy*。允許 Greengrass 核心裝置與 AWS IoT 和 通訊 AWS IoT 的政策名稱 AWS IoT Greengrass。如果 AWS IoT 政策不存在，安裝程式會使用此名稱建立寬鬆 AWS IoT 政策。您可以限制此政策的許可給您的使用案例。如需詳細資訊，請參閱[AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。
- g. *GreengrassV2TokenExchangeRole*。允許 Greengrass 核心裝置取得臨時 AWS 登入資料的 IAM 角色名稱。如果角色不存在，安裝程式會建立該角色，並建立和連接名為 的政策 *GreengrassV2TokenExchangeRoleAccess*。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。
- h. *GreengrassCoreTokenExchangeRoleAlias*。IAM 角色的別名，允許 Greengrass 核心裝置稍後取得臨時登入資料。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM 角色。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果成功，安裝程式會列印下列訊息：

- 如果您指定 `--provision`，安裝程式會在成功設定資源 `Successfully configured Nucleus with provisioned resource details` 時列印。
 - 如果您指定 `--deploy-dev-tools`，`Configured Nucleus to deploy aws.greengrass.Cli component` 安裝程式會在成功建立部署時列印。
 - 如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為服務 `Successfully set up Nucleus as a system service` 時列印。
 - 如果您未指定 `--setup-system-service true`，安裝程式 `Launched Nucleus successfully` 會在成功時列印並執行軟體。
2. 如果您安裝 v2 [Greengrass 核](#).0.4 或更新版本，請略過此步驟。如果您下載了最新版本的軟體，則已安裝 v2.0.4 或更新版本。

執行下列命令，為您的 AWS IoT Greengrass Core 軟體根資料夾設定所需的檔案許可。/
greengrass/v2 以您在安裝命令中指定的根資料夾取代，並以根資料夾的父資料夾取代 /
greengrass。

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)。

Note

根據預設，安裝程式建立的 IAM 角色不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須將許可新增至角色，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

如果您還沒有元件成品的 S3 儲存貯體，您可以在建立儲存貯體之後新增這些許可。

Note

當 AWS IoT Greengrass Core 軟體連線至雲端時，您的裝置將被識別為 Core 裝置。

如需如何設定和使用軟體和 的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體包含安裝程式，可將您的裝置設定為 Greengrass 核心裝置。若要手動設定裝置，您可以建立裝置要使用的必要 AWS IoT 和 IAM 資源。如果您手動建立這些資源，則不需要提供 AWS 登入資料給安裝程式。

當您手動安裝 AWS IoT Greengrass Core 軟體時，您也可以將裝置設定為使用網路代理或在連接埠 443 AWS 上連線至。例如，如果您的裝置在防火牆或網路代理之後執行，您可能需要指定這些組態選項。如需詳細資訊，請參閱[連線至連接埠 443 或透過網路代理](#)。

您也可以設定 AWS IoT Greengrass Core 軟體，透過 [PKCS#11 介面](#) 使用硬體安全模組 (HSM)。此功能可讓您安全地存放私有金鑰和憑證檔案，使其不會在軟體中公開或複製。您可以在硬體模組上存放私有金鑰和憑證，例如 HSM、信任平台模組 (TPM) 或其他密碼編譯元素。此功能僅適用於 Linux 裝置。如需硬體安全性和使用需求的詳細資訊，請參閱 [硬體安全整合](#)。

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [擷取 AWS IoT 端點](#)
- [建立 AWS IoT 物件](#)
- [建立物件憑證](#)
- [設定物件憑證](#)
- [建立權杖交換角色](#)
- [下載憑證至裝置](#)
- [設定裝置環境](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)

擷取 AWS IoT 端點

取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置會使用這些端點來連線 AWS IoT。請執行下列操作：

1. 取得 AWS IoT 的資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
```

```
}
```

2. 取得的 AWS IoT 登入資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

建立 AWS IoT 物件

AWS IoT 物件代表連線至 的裝置和邏輯實體 AWS IoT。Greengrass 核心裝置是 AWS IoT 實物。當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。

在本節中，您會建立代表裝置的 AWS IoT 物件。

建立 AWS IoT 物件

1. 為您的裝置建立 AWS IoT 物件。在您的開發電腦上，執行下列命令。
 - 將 *MyGreengrassCore* 取代為要使用的物件名稱。此名稱也是 Greengrass 核心裝置的名稱。

Note

物件名稱不能包含冒號 (:) 字元。

```
aws iot create-thing --thing-name MyGreengrassCore
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
```

```
}
```

2. (選用) 將 AWS IoT 物件新增至新的或現有的物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的機群。將軟體元件部署至裝置時，您可以將個別裝置或裝置群組設為目標。您可以將裝置新增至具有作用中 Greengrass 部署的物件群組，以將該物件群組的軟體元件部署到裝置。請執行下列操作：

- a. (選用) 建立 AWS IoT 物件群組。

- 將 *MyGreengrassCoreGroup* 取代為要建立的物件群組名稱。

 Note

物件群組名稱不能包含冒號 (:) 字元。

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. 將 AWS IoT 物件新增至物件群組。

- 將 *MyGreengrassCore* 取代為您 AWS IoT 物件的名稱。
- 將 *MyGreengrassCoreGroup* 取代為物件群組的名稱。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，命令不會有任何輸出。

建立物件憑證

當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。此憑證可讓裝置與 AWS IoT 和 通訊 AWS IoT Greengrass。

在本節中，您可以建立和下載裝置可用來連線的憑證 AWS。

如果您想要將 AWS IoT Greengrass Core 軟體設定為使用硬體安全模組 (HSM) 來安全地存放私有金鑰和憑證，請依照步驟從 HSM 中的私有金鑰建立憑證。否則，請依照步驟在 AWS IoT 服務中建立憑證和私有金鑰。硬體安全功能僅適用於 Linux 裝置。如需硬體安全性和使用需求的詳細資訊，請參閱 [硬體安全整合](#)。

在 AWS IoT 服務中建立憑證和私有金鑰

建立物件憑證

1. 建立資料夾以下載 AWS IoT 物件的憑證。

```
mkdir greengrass-v2-certs
```

2. 建立並下載 AWS IoT 物件的憑證。

```
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
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificateId":  
  "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificatePem": "-----BEGIN CERTIFICATE-----  
MIICiTCCAFICCCQD6m7oRw0uX0jANBgkqhkiG9w  
0BAQUFADCBiDELMakGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ  
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xZDASBgNVBAwTC01BTSBDb25zb2x1MRIw  
EAYDVQQDEw1UZXR0Q21sYWVhZAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvbi5  
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMakGA1UEBh  
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb  
WF6b24xZDASBgNVBAwTC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWVh  
ZAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvbi5jb20wZ8wDQYJKoZIhvcNAQE
```

```

BBQADgY0AMIGJAOGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPEQEFAAOCAQ8AMIIBCgKCAQEAEEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTWO
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEeahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEEv9mQ0UXP6p1fgxwKRX2fEXAMPLEda\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}

```

儲存憑證的 Amazon Resource Name (ARN)，以供稍後用於設定憑證。

從 HSM 中的私有金鑰建立憑證

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

建立物件憑證

1. 在核心裝置上，初始化 HSM 中的 PKCS#11 字符，並產生私有金鑰。私有金鑰必須是 RSA-2048 金鑰大小（或更大）或 ECC 金鑰的 RSA 金鑰。

 Note

若要搭配 ECC 金鑰使用硬體安全模組，您必須使用 [Greengrass nucleus](#) v2.5.6 或更新版本。

若要使用硬體安全模組和[秘密管理員](#)，您必須搭配 RSA 金鑰使用硬體安全模組。

檢查您的 HSM 文件，了解如何初始化字符並產生私有金鑰。如果您的 HSM 支援物件 IDs，請在產生私有金鑰時指定物件 ID。儲存您在初始化字符和產生私有金鑰時指定的槽 ID、使用者 PIN、物件標籤、物件 ID（如果您的 HSM 使用）。您稍後將物件憑證匯入 HSM 並設定 AWS IoT Greengrass 核心軟體時，會使用這些值。

2. 從私有金鑰建立憑證簽署請求 (CSR)。AWS IoT 使用此 CSR 為您在 HSM 中產生的私有金鑰建立物件憑證。如需有關如何從私有金鑰建立 CSR 的資訊，請參閱 HSM 的文件。CSR 是檔案，例如 `iotdevicekey.csr`。
3. 將 CSR 從裝置複製到您的開發電腦。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸 CSR。將 *device-ip-address* 取代為您裝置的 IP 地址，並將 *~/iotdevicekey.csr* 取代為裝置上的 CSR 檔案路徑。

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 在開發電腦上，建立資料夾以下載 AWS IoT 物件的憑證。

```
mkdir greengrass-v2-certs
```

5. 使用 CSR 檔案來建立 AWS IoT 物件的憑證，並將其下載到您的開發電腦。

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId": "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----"
```

```

MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBiDELMakGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBA5TC01BTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXR0Q21sYWMyH2AdBgkqhkiG9w0BCQEWEG5vb25lQGfYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMakGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBA5TC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWMy
H2AdBgkqhkiG9w0BCQEWEG5vb25lQGfYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLyGVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}

```

儲存憑證的 ARN，以供稍後用於設定憑證。

設定物件憑證

將物件憑證連接至您先前建立的 AWS IoT 物件，並將 AWS IoT 政策新增至憑證，以定義核心裝置的 AWS IoT 許可。

設定物件的憑證

1. 將憑證連接至 AWS IoT 物件。

- 將 *MyGreengrassCore* 取代為您 AWS IoT 物件的名稱。
- 將憑證 Amazon Resource Name (ARN) 取代為您在上一步驟中建立之憑證的 ARN。

```

aws iot attach-thing-principal --thing-name MyGreengrassCore
--principal arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4

```

如果請求成功，命令不會有任何輸出。

2. 建立並連接 AWS IoT 政策，以定義 Greengrass 核心裝置的 AWS IoT 許可。下列政策允許存取所有 MQTT 主題和 Greengrass 操作，因此您的裝置可以使用需要新 Greengrass 操作的自訂應

用程式和未來變更。您可以根據使用案例來限制此政策關閉。如需詳細資訊，請參閱[AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。

如果您之前已設定 Greengrass 核心裝置，您可以連接其 AWS IoT 政策，而不是建立新的政策。

請執行下列操作：

- a. 建立檔案，其中包含 Greengrass 核心裝置所需的 AWS IoT 政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano greengrass-v2-iot-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- b. 從 AWS IoT 政策文件建立政策。

- 將 *GreengrassV2IoTThingPolicy* 取代為要建立的政策名稱。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "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. 將 AWS IoT 政策連接至 AWS IoT 物件的憑證。

- 將 *GreengrassV2IoTThingPolicy* 取代為要連接的政策名稱。
- 將目標 ARN 取代為您 AWS IoT 物件的憑證 ARN。

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，命令不會有任何輸出。

建立權杖交換角色

Greengrass 核心裝置使用稱為字符交換角色的 IAM 服務角色來授權呼叫 AWS 服務。裝置使用 AWS IoT 登入資料提供者取得此角色的臨時 AWS 登入資料，允許裝置與 互動 AWS IoT、將日誌傳送至 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

您可以使用 AWS IoT 角色別名來設定 Greengrass 核心裝置的字符交換角色。角色別名可讓您變更裝置的字符交換角色，但保持裝置的組態相同。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權對 AWS 服務的直接呼叫](#)。

在本節中，您會建立權杖交換 IAM 角色和指向角色 AWS IoT 的角色別名。如果您已經設定 Greengrass 核心裝置，您可以使用其權杖交換角色和角色別名，而不是建立新的權杖交換角色和角色別名。然後，您將裝置的 AWS IoT 物件設定為使用該角色和別名。

建立權杖交換 IAM 角色

1. 建立您的裝置可用作字符交換角色的 IAM 角色。請執行下列操作：

- a. 建立檔案，其中包含權杖交換角色所需的信任政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-trust-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 使用信任政策文件建立權杖交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為要建立的 IAM 角色名稱。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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. 建立檔案，其中包含權杖交換角色所需的存取政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```



```

    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
    ],
    "Resource": "*"
  }
]
}

```

Note

此存取政策不允許存取 S3 儲存貯體中的元件成品。若要部署在 Amazon S3 中定義成品的自訂元件，您必須將許可新增至角色，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

如果您還沒有元件成品的 S3 儲存貯體，您可以在建立儲存貯體之後新增這些許可。

d. 從政策文件中建立 IAM 政策。

- 以要建立的 IAM 政策名稱取代 *GreengrassV2TokenExchangeRoleAccess*。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```

{
  "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. 將 IAM 政策連接至字符交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為 IAM 角色的名稱。
- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，命令不會有任何輸出。

2. 建立指向字符交換 AWS IoT 角色的角色別名。

- 以要建立的角色別名名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
- 將角色 ARN 取代為您在上一個步驟中建立之 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

若要建立角色別名，您必須具有傳遞字符交換 IAM 角色的許可 AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的 AWS 使用者是否具有此許可。如需詳細資訊，請參閱 AWS Identity and Access Management 《使用者指南》中的 [授予使用者將角色傳遞至 AWS 服務的許可](#)。

3. 建立並連接允許 Greengrass 核心裝置使用角色別名來擔任字符交換角色 AWS IoT 的政策。如果您之前已設定 Greengrass 核心裝置，您可以連接其角色別名 AWS IoT 政策，而不是建立新的。請執行下列操作：

- a. （選用）建立檔案，其中包含角色別名所需的 AWS IoT 政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano greengrass-v2-iot-role-alias-policy.json
```

將下列 JSON 複製到 檔案。

- 使用角色別名的 ARN 取代資源 ARN。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/  
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

- b. 從 AWS IoT 政策文件建立政策。

- 將 *GreengrassCoreTokenExchangeRoleAliasPolicy* 取代為要建立 AWS IoT 的政策名稱。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy  
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "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. 將 AWS IoT 政策連接至 AWS IoT 物件的憑證。

- 以角色別名 AWS IoT 政策的名稱取代 *GreengrassCoreTokenExchangeRoleAliasPolicy*。
- 將目標 ARN 取代為您 AWS IoT 物件的憑證 ARN。

```

aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4

```

如果請求成功，命令不會有任何輸出。

下載憑證至裝置

先前，您已將裝置的憑證下載到開發電腦。在本節中，您將憑證複製到核心裝置，以使用裝置用來連線的憑證來設定裝置 AWS IoT。您也可以下載 Amazon 根憑證授權機構 (CA) 憑證。如果您使用 HSM，您也可以將憑證檔案匯入本節的 HSM。

- 如果您先前已在 AWS IoT 服務中建立物件憑證和私有金鑰，請依照步驟下載具有私有金鑰和憑證檔案的憑證。
- 如果您先前從硬體安全模組 (HSM) 中的私有金鑰建立物件憑證，請依照步驟下載 HSM 中具有私有金鑰和憑證的憑證。

下載具有私有金鑰和憑證檔案的憑證

將憑證下載至裝置

1. 將 AWS IoT 物件憑證從開發電腦複製到裝置。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸憑證。將 *device-ip-address* 取代為您裝置的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 在裝置上建立 Greengrass 根資料夾。您稍後會將 AWS IoT Greengrass 核心軟體安裝到此資料夾。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 `C:\greengrass\v2` 或 `D:\greengrass\v2`，將 Greengrass 元件路徑保持在 260 個字元限制以下。

Linux or Unix

- */greengrass/v2* 將 取代為要使用的資料夾。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 *C#\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 *C#\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

3. (僅限 Linux) 設定 Greengrass 根資料夾父項的許可。

- 將 `/greengrass` 取代為根資料夾的父系。

```
sudo chmod 755 /greengrass
```

4. 將 AWS IoT 物件憑證複製到 Greengrass 根資料夾。

Linux or Unix

- `/greengrass/v2` 將 取代為 Greengrass 根資料夾。

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

Windows Command Prompt

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

PowerShell

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. 根據預設，下載 Amazon 根憑證授權機構 (CA) certificate. AWS IoT certificates 會與 Amazon 根憑證授權機構憑證相關聯。

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
```

在 HSM 中使用私有金鑰和憑證下載憑證

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

將憑證下載至裝置

1. 將 AWS IoT 物件憑證從開發電腦複製到裝置。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 scp 命令來傳輸憑證。將 *device-ip-address* 取代為您裝置的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 在裝置上建立 Greengrass 根資料夾。您稍後會將 AWS IoT Greengrass 核心軟體安裝到此資料夾。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 C:\greengrass\v2 或 D:\greengrass\v2，將 Greengrass 元件路徑保持在 260 個字元限制以下。

Linux or Unix

- */greengrass/v2* 將取代為要使用的資料夾。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

- (僅限 Linux) 設定 Greengrass 根資料夾父項的許可。

- 將 `/greengrass` 取代為根資料夾的父系。

```
sudo chmod 755 /greengrass
```

- 將物件憑證檔案 匯入 `~/greengrass-v2-certs/device.pem.crt`HSM。檢查 HSM 的文件，了解如何將憑證匯入其中。使用您先前在 HSM 中產生私有金鑰的相同字符、槽 ID、使用者 PIN、物件標籤和物件 ID (如果您的 HSM 使用) 匯入憑證。

Note

如果您先前在沒有物件 ID 的情況下產生私有金鑰，且憑證具有物件 ID，請將私有金鑰的物件 ID 設定為與憑證相同的值。檢查您的 HSM 文件，了解如何設定私有金鑰物件的物件 ID。

- (選用) 刪除物件憑證檔案，使其僅存在於 HSM 中。

```
rm ~/greengrass-v2-certs/device.pem.crt
```

- 根據預設，下載 Amazon 根憑證授權機構 (CA) certificate. AWS IoT certificates 會與 Amazon 根憑證授權機構憑證相關聯。

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
```

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成後，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. （選用）建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者（通常為 `root`）具有使用任何使用者和任何群組執行 `sudo` 的許可。
 - a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. （選用）若要[執行容器化 Lambda 函數](#)，您必須啟用 [cgrouops](#) v1，而且必須啟用和掛載記憶體和裝置 `cgrouops`。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 `cgrouops` 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱您作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 中的需求清單所示[裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.0 和更新版本。

為 設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
 4. 在 Windows 裝置的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是 [wmic 已棄用命令](#) 的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

- 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
- 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將 `##` 取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇以 Accept 同意授權並執行命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將##取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為的檔案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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#)之規定。

2. （選用）驗證 Greengrass 核軟體簽章

 Note

此功能適用於 Greengrass 核版本 2.9.5 及更新版本。

- a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 調用會產生輸出，指出驗證的結果。
- i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：

```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。

3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

如果您安裝的 Greengrass 核版本早於 2.4.0 版，安裝 AWS IoT Greengrass 核心軟體後請勿移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass Core 軟體後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從部分組態檔案安裝，指定使用您先前建立 AWS 的資源和憑證。Core AWS IoT Greengrass 軟體使用組態檔案，指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案建立完整的組態檔案。

- 指定 以使用 `ggc_user` 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 `ggc_group` 系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 [使用 JVM 選項控制記憶體配置](#)。

- 如果您先前已在 AWS IoT 服務中建立物件憑證和私有金鑰，請依照步驟安裝具有私有金鑰和憑證檔案的 AWS IoT Greengrass 核心軟體。
- 如果您先前從硬體安全模組 (HSM) 中的私有金鑰建立物件憑證，請依照步驟在 HSM 中安裝具有私有金鑰和憑證 AWS IoT Greengrass 的 Core 軟體。

使用私有金鑰和憑證檔案安裝 AWS IoT Greengrass 核心軟體

安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass Core 軟體的版本。

- 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。


```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到 檔案。此部分組態檔案會指定系統參數和 Greengrass 核參數。

```
---
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"
```

然後，執行下列動作：

- 將 的每個執行個體取代 */greengrass/v2* 為 Greengrass 根資料夾。
- 將 *MyGreengrassCore* 取代為物件的名稱 AWS IoT。
- 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
- 將 *us-west-2* 取代 AWS 區域 為您建立資源的。
- 以字符交換角色別名的名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
- 將 取代 *iotDataEndpoint* 為 AWS IoT 您的資料端點。
- 將 取代 *iotCredEndpoint* 為您的 AWS IoT 登入資料端點。

Note

在此組態檔案中，您可以自訂其他核組態選項，例如要使用的連接埠和網路代理，如下列範例所示。如需詳細資訊，請參閱 [Greengrass 核組態](#)。

```

---
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. 執行安裝程式，並指定 `--init-config` 以提供組態檔案。

- 將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 Greengrass 根資料夾。
- 將 `GreengrassInstaller` 的每個執行個體取代為您解壓縮安裝程式的資料夾。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需有關直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

4. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 config、packages 和 logs。

在 HSM 中使用私有金鑰和憑證安裝 AWS IoT Greengrass 核心軟體

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass Core 軟體的版本。

- 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 若要讓 AWS IoT Greengrass Core 軟體使用 HSM 中的私有金鑰和憑證，請在安裝 AWS IoT Greengrass Core 軟體時安裝 [PKCS#11 提供者元件](#)。PKCS#11 供應商元件是您可以在安裝期間設定的外掛程式。您可以從下列位置下載最新版本的 PKCS#11 提供者元件：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>

將 PKCS#11 供應商外掛程式下載至名為 的檔

案aws.greengrass.crypto.Pkcs11Provider.jar。將 *GreengrassInstaller* 取代為您要使用的資料夾。

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

3. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到 檔案。此部分組態檔案會指定系統參數、Greengrass 核參數和 PKCS#11 提供者參數。

```
---
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"
```

然後，執行下列動作：

- 將 PKCS#11 URIs 中的每個 *iotdevicekey* 執行個體取代為您建立私有金鑰並匯入憑證的物件標籤。
- 將 的每個執行個體取代 */greengrass/v2* 為 Greengrass 根資料夾。
- 將 *MyGreengrassCore* 取代為物件的名稱 AWS IoT 。
- 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
- 將 *us-west-2* 取代 AWS 區域 為您建立資源的 。
- 以字符交換角色別名的名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
- 將 取代 *iotDataEndpoint* 為 AWS IoT 您的資料端點。
- 將 取代 *iotCredEndpoint* 為您的 AWS IoT 登入資料端點。
- 將 *aws.greengrass.crypto.Pkcs11Provider* 元件的組態參數取代為核心裝置上的 HSM 組態值。

Note

在此組態檔案中，您可以自訂其他核組態選項，例如要使用的連接埠和網路代理，如下列範例所示。如需詳細資訊，請參閱 [Greengrass 核組態](#)。

```
---
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. 執行安裝程式，並指定 `--init-config` 以提供組態檔案。

- `/greengrass/v2` 將取代為 Greengrass 根資料夾。
- 將 `GreengrassInstaller` 的每個執行個體取代為您解壓縮安裝程式的資料夾。

```

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需有關直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

5. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 `config`、`packages` 和 `logs`。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱 [執行 AWS IoT Greengrass 核心軟體](#)。

Note

當 AWS IoT Greengrass Core 軟體連線至雲端時，您的裝置將被識別為 Core 裝置。

如需如何設定和使用軟體的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體

此功能適用於 [Greengrass 核心元件](#) 的 v2.4.0 和更新版本。

透過機 AWS IoT 群佈建，您可以設定 AWS IoT AWS IoT，在裝置第一次連線到時，產生並安全地將 X.509 裝置憑證和私有金鑰交付至您的裝置。AWS IoT 提供由 Amazon 根憑證授權單位 (CA) 簽署的用戶端憑證。您也可以設定 AWS IoT 來指定您使用機群佈建佈建的 Greengrass 核心裝置的物件群組、物件類型和許可。您可以定義佈建範本，以定義如何 AWS IoT 佈建每個裝置。佈建範本會指定佈建時要為裝置建立的物件、政策和憑證資源。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[佈建範本](#)。

AWS IoT Greengrass 提供機 AWS IoT 群佈建外掛程式，可讓您使用機 AWS IoT 群佈建建立 AWS 的資源來安裝 AWS IoT Greengrass Core 軟體。機群佈建外掛程式會依宣告使用佈建。裝置使用佈建宣告憑證和私有金鑰來取得可用於一般操作的唯一 X.509 裝置憑證和私有金鑰。您可以在製造期間將宣告憑證和私有金鑰嵌入每個裝置，讓您的客戶稍後可以在每個裝置上線時啟用裝置。您可以針對多個裝置使用相同的宣告憑證和私有金鑰。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[依宣告佈建](#)。

Note

機群佈建外掛程式目前不支援在硬體安全模組 (HSM) 中存放私有金鑰和憑證檔案。若要使用 HSM，[請使用手動佈建安裝 AWS IoT Greengrass 核心軟體](#)。

若要使用機 AWS IoT 群佈建安裝 AWS IoT Greengrass 核心軟體，您必須在 中設定 AWS IoT AWS 帳戶用來佈建 Greengrass 核心裝置的資源。這些資源包括佈建範本、宣告憑證和[字符交換 IAM 角色](#)。建立這些資源之後，您可以重複使用這些資源，在機群中佈建多個核心裝置。如需詳細資訊，請參閱[設定 Greengrass 核心裝置的 AWS IoT 機群佈建](#)。

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [先決條件](#)
- [擷取 AWS IoT 端點](#)

- [下載憑證至裝置](#)
- [設定裝置環境](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [下載 AWS IoT 機群佈建外掛程式](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)
- [設定 Greengrass 核心裝置的 AWS IoT 機群佈建](#)
- [設定 AWS IoT 機群佈建外掛程式](#)
- [AWS IoT 機群佈建外掛程式 changelog](#)

先決條件

若要使用機 AWS IoT 群佈建安裝 AWS IoT Greengrass 核心軟體，您必須先[為 Greengrass 核心裝置設定 AWS IoT 機群佈建](#)。完成這些步驟一次後，您可以使用機群佈建，在任何數量的裝置上安裝 AWS IoT Greengrass Core 軟體。

擷取 AWS IoT 端點

取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置會使用這些端點來連線 AWS IoT。請執行下列操作：

1. 取得 AWS IoT 的資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 取得的 AWS IoT 登入資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

如果請求成功，回應看起來與下列範例類似。

```
{
```

```
"endpointAddress": "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
}
```

下載憑證至裝置

裝置使用宣告憑證和私有金鑰來驗證其佈建 AWS 資源和取得 X.509 裝置憑證的請求。您可以在製造期間將宣告憑證和私有金鑰嵌入裝置，或在安裝期間將憑證和金鑰複製到裝置。在本節中，您將宣告憑證和私有金鑰複製到裝置。您也可以將 Amazon 根憑證授權機構 (CA) 憑證下載到裝置。

Important

佈建宣告私有金鑰應隨時受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控是否有濫用跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到濫用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[監控 AWS IoT](#)。

為了協助您更有效管理裝置數量，以及在中自行註冊的裝置 AWS 帳戶，您可以在建立機群佈建範本時指定預先佈建掛鉤。預先佈建掛鉤是一種 AWS Lambda 函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建掛鉤，檢查資料庫的裝置 ID，以確認裝置具有佈建的許可。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[預先佈建掛鉤](#)。

將宣告憑證下載至裝置

1. 將宣告憑證和私有金鑰複製到裝置。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸宣告憑證和私有金鑰。下列範例命令 `claim-certs` 會將開發電腦上名為的資料夾傳輸至裝置。將 *device-ip-address* 取代為您裝置的 IP 地址。

```
scp -r claim-certs/ device-ip-address:~
```

2. 在裝置上建立 Greengrass 根資料夾。您稍後會將 AWS IoT Greengrass 核心軟體安裝到此資料夾。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 `C:\greengrass\v2` 或 `D:\greengrass\v2`，將 Greengrass 元件路徑保持在 260 個字元限制以下。

Linux or Unix

- `/greengrass/v2` 將 取代為要使用的資料夾。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

3. (僅限 Linux) 設定 Greengrass 根資料夾父項的許可。

- 將 `/greengrass` 取代為根資料夾的父系。

```
sudo chmod 755 /greengrass
```

4. 將宣告憑證移至 Greengrass 根資料夾。

- 將 `/greengrass/v2`或 `C#\greengrass\v2` 取代為 Greengrass 根資料夾。

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. 根據預設，下載 Amazon 根憑證授權機構 (CA) certificate. AWS IoT certificates 會與 Amazon 根憑證授權機構憑證相關聯。

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
```

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. （選用）建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者（通常為 `root`）具有使用任何使用者和任何群組執行 `sudo` 的許可。

- a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. (選用) 若要[執行容器化 Lambda 函數](#)，您必須啟用 [cgroups](#) v1，而且必須啟用和掛載記憶體和裝置 cgroups。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 [中的需求清單](#)所示[裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核元件](#)的 v2.5.0 和更新版本。

為 設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是[wmic已棄用命令](#)的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
6. 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將##取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 以Accept同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass 核心軟體。將##取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為的檔案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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

2. (選用) 驗證 Greengrass 核軟體簽章

Note

此功能適用於 Greengrass 核版本 2.9.5 及更新版本。

a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -verify -certs -verbose greengrass-nucleus-latest.zip
```

b. jarsigner 調用會產生輸出，指出驗證的結果。

i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：

```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。
3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

如果您安裝的 Greengrass 核版本早於 2.4.0 版，請勿在安裝 AWS IoT Greengrass 核心軟體之後移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass Core 軟體後移除此資料夾。

下載 AWS IoT 機群佈建外掛程式

您可以從下列位置下載最新版本的 AWS IoT 機群佈建外掛程式：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar>

Note

您可以從下列位置下載特定版本的 AWS IoT 機群佈建外掛程式。將##取代為要下載的版本。如需每個機群佈建外掛程式版本的詳細資訊，請參閱 [AWS IoT 機群佈建外掛程式 changelog](#)。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

機群佈建外掛程式是開放原始碼。若要檢視其原始程式碼，請參閱 GitHub 上的 [AWS IoT 機群佈建外掛程式](#)。

下載 AWS IoT 機群佈建外掛程式

- 在您的裝置上，將 AWS IoT 機群佈建外掛程式下載至名為 的檔案aws.greengrass.FleetProvisioningByClaim.jar。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從指定使用機群佈建外掛程式佈建 AWS 資源的部分組態檔案安裝。Core AWS IoT Greengrass 軟體使用組態檔案，指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案和機群佈建外掛程式建立 AWS 的資源建立完整的組態檔案。
- 指定以使用 `ggc_user` 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 `ggc_group` 系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 [使用 JVM 選項控制記憶體配置](#)。

安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass Core 軟體的版本。

- 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到 檔案。此部分組態檔案會指定機群佈建外掛程式的參數。如需您可以指定之選項的詳細資訊，請參閱 [設定 AWS IoT 機群佈建外掛程式](#)。

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"
```

然後，執行下列動作：

- 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
- 將 */greengrass/v2* 或 *C#\greengrass\v2* 的每個執行個體取代為 Greengrass 根資料夾。

Note

在 Windows 裝置上，您必須將路徑分隔符號指定為雙反斜線 (\\)，例如 C:\\greengrass\\v2。

- 將 *us-west-2* 取代為您建立佈建範本和其他資源 AWS 的區域。
- 將 *iotDataEndpoint* 取代為您的 AWS IoT 資料端點。
- 將 *iotCredentialEndpoint* 取代為您的 AWS IoT 登入資料端點。
- 以字符交換角色別名的名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
- 以機群佈建範本的名稱取代 *GreengrassFleetProvisioningTemplate*。
- 將 *claimCertificatePath* 取代為裝置上的宣告憑證路徑。
- 將 *claimCertificatePrivateKeyPath* 取代為裝置上的宣告憑證私有金鑰路徑。
- 將範本參數 (templateParameters) 取代為用來佈建裝置的值。此範例是指定義 ThingName 和 ThingGroupName 參數的 [範例範本](#)。

 Note

在此組態檔案中，您可以自訂其他組態選項，例如要使用的連接埠和網路代理，如下列範例所示。如需詳細資訊，請參閱 [Greengrass 核組態](#)。

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"
```

若要使用 HTTPS 代理，您必須使用 1.1.0 版或更新版本的機群佈建外掛程式。您必須另外在 `rootCaPath` 下指定 `system`，如下列範例所示。

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. 執行安裝程式。指定 `--trusted-plugin` 以提供機群佈建外掛程式，並指定 `--init-config` 提供組態檔案。
 - */greengrass/v2* 將取代為 Greengrass 根資料夾。
 - 將 *GreengrassInstaller* 的每個執行個體取代為您解壓縮安裝程式的資料夾。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

4. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 config、packages 和 logs。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)。

如需如何設定和使用軟體和 的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

設定 Greengrass 核心裝置的 AWS IoT 機群佈建

若要[使用機群佈建安裝 AWS IoT Greengrass 核心軟體](#)，您必須先在 中設定下列資源 AWS 帳戶。這些資源可讓裝置向 註冊，AWS IoT 並以 Greengrass 核心裝置的形式運作。請遵循本節中的步驟，在 中建立和設定這些資源 AWS 帳戶。

- 權杖交換 IAM 角色，核心裝置會使用該角色來授權對 AWS 服務的呼叫。
- 指向字符交換角色 AWS IoT 的角色別名。
- (選用) 政策，核心裝置會使用該 AWS IoT 政策來授權對 AWS IoT 和 AWS IoT Greengrass 服務的呼叫。此 AWS IoT 政策必須允許指向字符交換 AWS IoT 角色的角色別名 `iot:AssumeRoleWithCertificate` 許可。

您可以針對機群中的所有核心裝置使用單一 AWS IoT 政策，也可以設定機群佈建範本來為每個核心裝置建立 AWS IoT 政策。

- AWS IoT 機群佈建範本。此範本必須指定下列項目：

- 物件 AWS IoT 資源。您可以指定現有物件群組的清單，以便在元件上線時將元件部署到每個裝置。
- AWS IoT 政策資源。此資源可以定義下列其中一個屬性：
 - 現有 AWS IoT 政策的名稱。如果您選擇此選項，則您從此範本建立的核心裝置會使用相同的 AWS IoT 政策，而且您可以以機群身分管理其許可。
 - AWS IoT 政策文件。如果您選擇此選項，您從此範本建立的每個核心裝置都會使用唯一的 AWS IoT 政策，而且您可以管理每個個別核心裝置的許可。
- AWS IoT 憑證資源。此憑證資源必須使用 `AWS::IoT::Certificate::Id` 參數將憑證連接至核心裝置。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[Just-in-time 佈建](#)。
- 機群 AWS IoT 佈建範本的佈建宣告憑證和私有金鑰。您可以在製造期間將此憑證和私有金鑰嵌入裝置，讓裝置可以在上線時自行註冊和佈建。

Important

佈建宣告私有金鑰應隨時受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控是否有濫用跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到濫用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[監控 AWS IoT](#)。

為了協助您更有效管理在中自行註冊的裝置數量和裝置 AWS 帳戶，您可以在建立機群佈建範本時指定預先佈建掛鉤。預先佈建掛鉤是一種 AWS Lambda 函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建掛鉤，檢查資料庫的裝置 ID，以確認裝置具有佈建的許可。如需詳細資訊，請參閱《[開發人員指南](#)》中的[預先佈建掛鉤](#)。AWS IoT Core

- 您連接到佈建宣告憑證 AWS IoT 的政策，允許裝置註冊和使用機群佈建範本。

主題

- [建立權杖交換角色](#)
- [建立 AWS IoT 政策](#)
- [建立機群佈建範本](#)
- [建立佈建宣告憑證和私有金鑰](#)

建立權杖交換角色

Greengrass 核心裝置使用稱為字符交換角色的 IAM 服務角色來授權呼叫 AWS 服務。裝置使用 AWS IoT 登入資料提供者取得此角色的臨時 AWS 登入資料，允許裝置與 互動 AWS IoT、將日誌傳送至 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

您可以使用 AWS IoT 角色別名來設定 Greengrass 核心裝置的字符交換角色。角色別名可讓您變更裝置的字符交換角色，但保持裝置的組態相同。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權直接呼叫 AWS 服務](#)。

在本節中，您會建立權杖交換 IAM 角色和指向角色 AWS IoT 的角色別名。如果您已經設定 Greengrass 核心裝置，您可以使用其權杖交換角色和角色別名，而不是建立新的權杖交換角色和角色別名。

建立字符交換 IAM 角色

1. 建立您的裝置可用作字符交換角色的 IAM 角色。請執行下列操作：

- a. 建立檔案，其中包含權杖交換角色所需的信任政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-trust-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 使用信任政策文件建立權杖交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為要建立的 IAM 角色名稱。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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. 建立檔案，其中包含字符交換角色所需的存取政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
    ],
    "Resource": "*"
  }
]
}

```

Note

此存取政策不允許存取 S3 儲存貯體中的元件成品。若要部署定義 Amazon S3 中成品的自訂元件，您必須將許可新增至角色，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

如果您還沒有元件成品的 S3 儲存貯體，您可以在建立儲存貯體之後新增這些許可。

d. 從政策文件中建立 IAM 政策。

- 將 *GreengrassV2TokenExchangeRoleAccess* 取代為要建立的 IAM 政策名稱。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```

{
  "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. 將 IAM 政策連接至字符交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為 IAM 角色的名稱。
- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，命令不會有任何輸出。

2. 建立指向字符交換 AWS IoT 角色的角色別名。

- 以要建立的角色別名名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
- 將角色 ARN 取代為您在上一個步驟中建立之 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

若要建立角色別名，您必須具有傳遞字符交換 IAM 角色的許可 AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的 AWS 使用者是否具有此許可。如需詳細資訊，請參閱 AWS Identity and Access Management 《使用者指南》中的 [授予使用者將角色傳遞至 AWS 服務的許可](#)。

建立 AWS IoT 政策

將裝置註冊為 AWS IoT 物件後，該裝置可以使用數位憑證進行身分驗證 AWS。此憑證包含一或多個政策，這些 AWS IoT 政策定義裝置可搭配憑證使用的許可。這些政策允許裝置與 AWS IoT 和 通訊 AWS IoT Greengrass。

透過機 AWS IoT 群佈建，裝置會連線至 AWS IoT 以建立和下載裝置憑證。在下一個區段中建立的機群佈建範本中，您可以指定是否將相同的 AWS IoT 政策 AWS IoT 連接到所有裝置的憑證，或為每個裝置建立新的政策。

在本節中，您會建立 AWS IoT 連接到所有裝置憑證 AWS IoT 的政策。透過此方法，您可以將所有裝置的許可管理為機群。如果您想要為每個裝置建立新的 AWS IoT 政策，您可以略過本節，並在定義機群範本時參考其中的政策。

建立 AWS IoT 政策

- 建立 AWS IoT 政策來定義 Greengrass 核心裝置機群的 AWS IoT 許可。下列政策允許存取所有 MQTT 主題和 Greengrass 操作，因此您的裝置可以使用需要新 Greengrass 操作的自訂應用程式和未來變更。此政策也允許 `iot:AssumeRoleWithCertificate` 許可，允許您的裝置使用您在上一節中建立的字符交換角色。您可以根據使用案例來限制此政策關閉。如需詳細資訊，請參閱 [AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。

請執行下列操作：

- a. 建立檔案，其中包含 Greengrass 核心裝置所需的 AWS IoT 政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano greengrass-v2-iot-policy.json
```

將下列 JSON 複製到 檔案。

- 以您在上一節中建立 AWS IoT 的角色別名 ARN 取代 `iot:AssumeRoleWithCertificate` 資源。

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. 從 AWS IoT 政策文件中建立政策。

- 將 *GreengrassV2IoTThingPolicy* 取代為要建立的政策名稱。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果請求成功，回應看起來與下列範例類似。

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"
}

```

建立機群佈建範本

AWS IoT 機群佈建範本定義如何佈建 AWS IoT 物件、政策和憑證。若要使用機群佈建外掛程式佈建 Greengrass 核心裝置，您必須建立指定下列項目的範本：

- 物件 AWS IoT 資源。您可以指定現有物件群組的清單，以便在元件上線時將元件部署到每個裝置。
- AWS IoT 政策資源。此資源可以定義下列其中一個屬性：
 - 現有 AWS IoT 政策的名稱。如果您選擇此選項，您從此範本建立的核心裝置會使用相同的 AWS IoT 政策，而且您可以將其許可做為機群管理。
 - AWS IoT 政策文件。如果您選擇此選項，您從此範本建立的每個核心裝置都會使用唯一的 AWS IoT 政策，而且您可以管理每個個別核心裝置的許可。
- AWS IoT 憑證資源。此憑證資源必須使用 `AWS::IoT::Certificate::Id` 參數將憑證連接至核心裝置。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 [Just-in-time 佈建](#)。

在範本中，您可以指定將 AWS IoT 物件新增至現有物件群組的清單。當核心裝置 AWS IoT Greengrass 第一次連線至時，它會針對其成員所在的每個物件群組接收 Greengrass 部署。您可以使用物件群組，在裝置上線後立即將最新的軟體部署到每個裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

在佈建裝置 AWS 帳戶時，AWS IoT 服務需要許可，才能在中建立和更新 AWS IoT 資源。若要授予 AWS IoT 服務存取權，您可以建立 IAM 角色，並在建立範本時提供該角色。AWS IoT 提供受管政策 [AWSIoTThingsRegistration](#)，允許存取佈建裝置時 AWS IoT 可能使用的所有許可。您可以使用此受管政策，或建立自訂政策，針對您的使用案例縮小受管政策中的許可範圍。

在本節中，您會建立允許 AWS IoT 佈建裝置資源的 IAM 角色，並建立使用該 IAM 角色的機群佈建範本。

建立機群佈建範本

1. 建立 AWS IoT 可在中佈建資源的 IAM 角色 AWS 帳戶。請執行下列操作：

- a. 建立檔案，其中包含允許 AWS IoT 擔任角色的信任政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano aws-iot-trust-policy.json
```

將下列 JSON 複製到檔案。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 使用信任政策文件建立 IAM 角色。

- 將 *GreengrassFleetProvisioningRole* 取代為要建立的 IAM 角色名稱。

```
aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-policy-document file://aws-iot-trust-policy.json
```

如果請求成功，回應看起來與下列範例類似。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": "*"
    }
  ]
}
```

- 檢閱 [AWSIoTThingsRegistration](#) 政策，允許存取佈建裝置時 AWS IoT 可能使用的所有許可。您可以使用此受管政策，或建立自訂政策，為您的使用案例定義縮小範圍的許可。如果您選擇建立自訂政策，請現在執行此操作。
- 將 IAM 政策連接至機群佈建角色。
 - 將 *GreengrassFleetProvisioningRole* 取代為 IAM 角色的名稱。
 - 如果您在上一個步驟中建立了自訂政策，請將政策 ARN 取代為要使用的 IAM 政策的 ARN。

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --  
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

如果請求成功，命令不會有任何輸出。

2. (選用) 建立預先佈建掛鉤，這是 AWS Lambda 驗證裝置在註冊期間提供之範本參數的函數。您可以使用預先佈建掛鉤，進一步控制您的 中加入哪些裝置以及裝置數量 AWS 帳戶。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[預先佈建掛鉤](#)。
3. 建立機群佈建範本。請執行下列操作：
 - a. 建立 檔案以包含佈建範本文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano greengrass-fleet-provisioning-template.json
```

撰寫佈建範本文件。您可以從下列範例佈建範本開始，該範本指定 建立具有下列屬性的 AWS IoT 物件：

- 物件的名稱是您在ThingName範本參數中指定的值。
- 物件是您在ThingGroupName範本參數中指定之物件群組的成員。物件群組必須存在於您的 中 AWS 帳戶。
- 物件的憑證已GreengrassV2IoTThingPolicy附加名為 AWS IoT 的政策。

如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[佈建範本](#)。

```
{  
  "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

MyThing、*MyPolicy* 和 *MyCertificate* 是任意名稱，可識別機群佈建範本中的每個資源規格。AWS IoT 不會在從範本建立的資源中使用這些名稱。您可以使用這些名稱，或將其取代為可協助您識別範本中每個資源的值。

b. 從佈建範本文件建立機群佈建範本。

- 將 *GreengrassFleetProvisioningTemplate* 取代為要建立的範本名稱。
- 將範本描述取代為您的範本描述。
- 將佈建角色 ARN 取代為您先前建立之角色的 ARN。

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

如果您建立了預先佈建掛鉤，請使用 `--pre-provisioning-hook` 引數指定預先佈建掛鉤 Lambda 函數的 ARN。

```
--pre-provisioning-hook targetArn=arn:aws:lambda:us-west-2:123456789012:function:GreengrassPreProvisioningHook
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
  "templateName": "GreengrassFleetProvisioningTemplate",
  "defaultVersionId": 1
}
```

建立佈建宣告憑證和私有金鑰

宣告憑證是 X.509 憑證，允許裝置註冊為 AWS IoT 實物，並擷取唯一 X.509 裝置憑證以用於一般操作。建立宣告憑證後，您可以連接 AWS IoT 政策，允許裝置使用它來建立唯一的裝置憑證，並使用機群佈建範本進行佈建。具有宣告憑證的裝置只能使用您在政策中 AWS IoT 允許的佈建範本進行佈建。

在本節中，您會建立宣告憑證，並將其設定為讓裝置與您在上一節中建立的機群佈建範本搭配使用。

⚠ Important

佈建宣告私有金鑰應隨時受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控是否有濫用跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到濫用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[監控 AWS IoT](#)。

為了協助您更有效管理裝置數量，以及在中自行註冊的裝置 AWS 帳戶，您可以在建立機群佈建範本時指定預先佈建掛鉤。預先佈建掛鉤是一種 AWS Lambda 函數，可驗證裝置在註冊期

間提供的範本參數。例如，您可以建立預先佈建掛鉤，檢查資料庫的裝置 ID，以確認裝置具有佈建的許可。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[預先佈建掛鉤](#)。

建立佈建宣告憑證和私有金鑰

1. 建立資料夾以下載宣告憑證和私有金鑰。

```
mkdir claim-certs
```

2. 建立並儲存憑證和私有金鑰以用於佈建。AWS IoT 提供由 Amazon 根憑證授權機構 (CA) 簽署的用戶端憑證。

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
```

如果請求成功，回應會包含憑證的相關資訊。儲存憑證的 ARN 以供稍後使用。

3. 建立並連接 AWS IoT 政策，允許裝置使用憑證來建立唯一的裝置憑證，並使用機群佈建範本進行佈建。下列政策允許存取裝置佈建 MQTT API。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[裝置佈建 MQTT API](#)。

請執行下列操作：

- a. 建立檔案，其中包含 Greengrass 核心裝置所需的 AWS IoT 政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano greengrass-provisioning-claim-iot-policy.json
```

將下列 JSON 複製到 檔案。

- 將每個##執行個體取代為您設定機群佈建 AWS 區域 的。
- 將 *account-id* 的每個執行個體取代為您的 AWS 帳戶 ID。
- 以您在上一節中建立的機群佈建範本名稱取代 *GreengrassFleetProvisioningTemplate* 的每個執行個體。

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. 從 AWS IoT 政策文件建立政策。

- 將 *GreengrassProvisioningClaimPolicy* 取代為要建立的政策名稱。

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

如果請求成功，回應看起來與下列範例類似。

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. 將 AWS IoT 政策連接至佈建宣告憑證。

- 將 *GreengrassProvisioningClaimPolicy* 取代為要連接的政策名稱。
- 將目標 ARN 取代為佈建宣告憑證的 ARN。

```

aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4

```

如果請求成功，命令不會有任何輸出。

您現在擁有佈建宣告憑證和私有金鑰，裝置可用來向註冊 AWS IoT，並將自己佈建為 Greengrass 核心裝置。您可以在製造期間將宣告憑證和私有金鑰嵌入裝置，或在安裝 AWS IoT Greengrass 核心軟體之前將憑證和金鑰複製到裝置。如需詳細資訊，請參閱[使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體](#)。

設定 AWS IoT 機群佈建外掛程式

AWS IoT 機群佈建外掛程式提供下列組態參數，您可以在[使用機群佈建安裝 AWS IoT Greengrass Core 軟體](#)時自訂這些參數。

`rootPath`

要做為 AWS IoT Greengrass Core 軟體根目錄的資料夾路徑。

`awsRegion`

機群佈建外掛程式用來佈建 AWS 資源 AWS 區域的。

`iotDataEndpoint`

您 AWS IoT 的資料端點 AWS 帳戶。

`iotCredentialEndpoint`

您的 AWS IoT 登入資料端點 AWS 帳戶。

`iotRoleAlias`

指向字符交換 IAM AWS IoT 角色的角色別名。AWS IoT 登入資料提供者會擔任此角色，以允許 Greengrass 核心裝置與 AWS 服務互動。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

`provisioningTemplate`

用於佈建 AWS 資源的 AWS IoT 機群佈建範本。此範本必須指定下列項目：

- AWS IoT 物件資源。您可以指定現有物件群組的清單，以便在元件上線時將元件部署到每個裝置。
- AWS IoT 政策資源。此資源可以定義下列其中一個屬性：
 - 現有 AWS IoT 政策的名稱。如果您選擇此選項，您從此範本建立的核心裝置會使用相同的 AWS IoT 政策，而且您可以將其許可做為機群管理。
 - AWS IoT 政策文件。如果您選擇此選項，您從此範本建立的每個核心裝置都會使用唯一的 AWS IoT 政策，而且您可以管理每個個別核心裝置的許可。
- AWS IoT 憑證資源。此憑證資源必須使用 `AWS::IoT::Certificate::Id` 參數將憑證連接至核心裝置。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[Just-in-time 佈建](#)。

如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[佈建範本](#)。

`claimCertificatePath`

您在 `provisioningTemplate` 中指定之佈建範本的佈建宣告憑證路徑。如需詳細資訊，請參閱 AWS IoT Core API 參考中的 [CreateProvisioningClaim](#)。

claimCertificatePrivateKeyPath

您在 中指定的佈建範本之佈建宣告憑證私有金鑰的路徑provisioningTemplate。如需詳細資訊，請參閱 AWS IoT Core API 參考中的 [CreateProvisioningClaim](#)。

Important

佈建宣告私有金鑰應隨時受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控是否有濫用跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到濫用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[監控 AWS IoT](#)。

為了協助您更有效管理裝置數量，以及在 中自行註冊的裝置 AWS 帳戶，您可以在建立機群佈建範本時指定預先佈建掛鉤。預先佈建掛鉤是一種 AWS Lambda 函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建掛鉤，檢查資料庫的裝置 ID，以確認裝置具有佈建的許可。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[預先佈建掛鉤](#)。

rootCaPath

Amazon 根憑證授權機構 (CA) 憑證的路徑。

templateParameters

(選用) 要提供給機群佈建範本的參數映射。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[佈建範本的參數一節](#)。

deviceId

(選用) 當機群佈建外掛程式建立 MQTT 連線時，用作用戶端 ID 的裝置識別符 AWS IoT。

預設：隨機 UUID。

mqttPort

(選用) 用於 MQTT 連線的連接埠。

預設：8883

proxyUrl

(選用) 代理伺服器的 URL，格式為 scheme://userinfo@host:port。若要使用 HTTPS 代理，您必須使用 1.1.0 版或更新版本的機群佈建外掛程式。

- `scheme` – 結構描述，必須是 `http` 或 `https`。

 Important

Greengrass 核心裝置必須執行 [Greengrass 核 v2.5.0](#) 或更新版本，才能使用 HTTPS 代理。

如果您設定 HTTPS 代理，則必須將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證。如需詳細資訊，請參閱[讓核心裝置信任 HTTPS 代理](#)。

- `userinfo` – (選用) 使用者名稱和密碼資訊。如果您在 `url` 中指定此資訊，Greengrass 核心裝置會忽略 `username` 和 `password` 欄位。
- `host` – 代理伺服器的主機名稱或 IP 地址。
- `port` – (選用) 連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：
 - `http` – 80
 - `https` – 443

`proxyUserName`

(選用) 驗證代理伺服器的使用者名稱。

`proxyPassword`

(選用) 驗證代理伺服器的使用者名稱。

`csrPath`

(選用) 憑證簽署請求 (CSR) 檔案的路徑，用於從 CSR 建立裝置憑證。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[依宣告佈建](#)。

`csrPrivateKeyPath`

(選用，如果宣告 `csrPath` 則為必要) 用於產生 CSR 的私有金鑰路徑。私有金鑰必須已用於產生 CSR。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[依宣告佈建](#)。

`certificatePath`

(選用) 用來儲存下載裝置憑證的路徑。

`privateKeyPath`

(選用) 用來儲存下載裝置私有金鑰的路徑。

AWS IoT 機群佈建外掛程式 changelog

下表說明透過宣告外掛程式 () 進行機 AWS IoT 群佈建的每個版本變更 `aws.greengrass.FleetProvisioningByClaim`。

版本	變更
1.2.2	錯誤修正和改善 • 新增對自訂核心裝置憑證 (<code>certificatePath</code>) 和私有金鑰 (<code>privateKeyPath</code>) 路徑的支援。
1.2.1	錯誤修正和改善 • 修正 Greengrass 核啟動期間機群佈建外掛程式離線的問題。機群佈建外掛程式現在會無限期重試 MQTT 連線呼叫。
1.2.0	錯誤修正和改善 • 新增透過憑證簽署請求搭配可設定私有金鑰路徑的裝置佈建支援。 • 次要修正和改善。
1.1.0	錯誤修正和改善 • 在 Windows 裝置上設定外掛程式時，新增對其他檔案路徑格式的支援。 • 新增對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱 連線至連接埠 443 或透過網路代理 和 讓核心裝置信任 HTTPS 代理。
1.0.0	初始版本。

使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體

此功能適用於 [Greengrass 核元件](#) 的 v2.4.0 和更新版本。

Core AWS IoT Greengrass 軟體安裝程式提供 Java 界面，您可以在佈建必要 AWS 資源的自訂外掛程式中實作此界面。您可以開發佈建外掛程式來使用自訂 X.509 用戶端憑證，或執行其他安裝程序不支援的複雜佈建步驟。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [建立您自己的用戶端憑證](#)。

若要在安裝 AWS IoT Greengrass Core 軟體時執行自訂佈建外掛程式，您可以建立您提供給安裝程式的 JAR 檔案。安裝程式會執行外掛程式，而外掛程式會傳回佈建組態，以定義 Greengrass 核心裝

置 AWS 的資源。安裝程式會使用此資訊在裝置上設定 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱[開發自訂佈建外掛程式](#)。

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [先決條件](#)
- [設定裝置環境](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)
- [開發自訂佈建外掛程式](#)

先決條件

若要使用自訂佈建安裝 AWS IoT Greengrass 核心軟體，您必須具備下列項目：

- 實作之自訂佈建外掛程式的 JAR 檔案 `DeviceIdentityInterface`。自訂佈建外掛程式必須傳回每個系統和 nucleus 組態參數的值。否則，您必須在安裝期間在組態檔案中提供這些值。如需詳細資訊，請參閱[開發自訂佈建外掛程式](#)。

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。
 - 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. （選用）建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者（通常為 `root`）具有使用任何使用者和任何群組執行 `sudo` 的許可。

- a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. (選用) 若要[執行容器化 Lambda 函數](#)，您必須啟用 [cgroups](#) v1，而且必須啟用和掛載記憶體和裝置 cgroups。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 [中的需求清單](#)所示[裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核元件](#)的 v2.5.0 和更新版本。

為 設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是[wmic已棄用命令](#)的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
6. 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將##取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 以Accept同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass 核心軟體。將##取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為的檔案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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

2. (選用) 驗證 Greengrass 核軟體簽章

Note

此功能適用於 Greengrass 核版本 2.9.5 和更新版本。

a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. jarsigner 調用會產生輸出，指出驗證的結果。

i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：


```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。
3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

如果您安裝比 v2.4.0 更舊的 Greengrass 核版本，請勿在安裝 AWS IoT Greengrass 核心軟體之後移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass 核心軟體後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從指定的部分組態檔案安裝，以使用您的自訂佈建外掛程式來佈建 AWS 資源。Core AWS IoT Greengrass 軟體使用組態檔案，指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案和自訂佈建外掛程式建立 AWS 的資源建立完整的組態檔案。
- 指定以使用 `ggc_user` 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 `ggc_group` 系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 [使用 JVM 選項控制記憶體配置](#)。

安裝 AWS IoT Greengrass Core 軟體 (Linux)

1. 檢查 AWS IoT Greengrass Core 軟體的版本。

- 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到 檔案。

```
---
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: ""
```

然後，執行下列動作：

- 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
- 將 的每個執行個體取代 *"/greengrass/v2"* 為 Greengrass 根資料夾。
- (選用) 指定系統和核組態值。如果您的佈建外掛程式不提供這些值，則必須設定這些值。
- (選用) 指定要提供給佈建外掛程式的組態參數。

Note

在此組態檔案中，您可以自訂其他組態選項，例如要使用的連接埠和網路代理，如下列範例所示。如需詳細資訊，請參閱 [Greengrass 核組態](#)。

```
---
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. 執行安裝程式。指定 `--trusted-plugin` 以提供您的自訂佈建外掛程式，並指定 `--init-config` 提供組態檔案。

 Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 C:\greengrass\v2 或 D:\greengrass\v2，將 Greengrass 元件路徑保持在 260 個字元限制以下。

- 將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 Greengrass 根資料夾。
- 將 `GreengrassInstaller` 的每個執行個體取代為您解壓縮安裝程式的資料夾。
- 將自訂佈建外掛程式 JAR 檔案的路徑取代為外掛程式 JAR 檔案的路徑。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

ℹ Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需有關直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

4. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 `config`、`packages` 和 `logs`。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱 [執行 AWS IoT Greengrass 核心軟體](#)。

如需如何設定和使用軟體的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

開發自訂佈建外掛程式

若要開發自訂佈建外掛程式，請建立實

作`com.aws.greengrass.provisioning.DeviceIdentityInterface`界面的 Java 類別。您可以在專案中包含 Greengrass 核 JAR 檔案，以存取此界面及其類別。此界面會定義輸入外掛程式組態並輸出佈建組態的方法。佈建組態會定義系統和 [Greengrass 核元件](#)的組態。Core AWS IoT Greengrass 軟體安裝程式使用此佈建組態在裝置上設定 AWS IoT Greengrass Core 軟體。

在您開發自訂佈建外掛程式之後，將其建置為 JAR 檔案，您可以提供給 AWS IoT Greengrass Core 軟體安裝程式，以在安裝期間執行外掛程式。安裝程式會在安裝程式使用的相同 JVM 中執行自訂佈建外掛程式，因此您可以建立僅包含外掛程式程式碼的 JAR。

Note

[AWS IoT 機群佈建外掛程式](#)實作 `DeviceIdentityInterface`，以在安裝期間使用機群佈建。機群佈建外掛程式是開放原始碼，因此您可以探索其原始程式碼，以查看如何使用佈建外掛程式界面的範例。如需詳細資訊，請參閱 GitHub 上的[AWS IoT 機群佈建外掛程式](#)。

主題

- [要求](#)
- [實作 `DeviceIdentityInterface` 介面](#)

要求

若要開發自訂佈建外掛程式，您必須建立符合下列要求的 Java 類別：

- 使用`com.aws.greengrass`套件或套件內的`com.aws.greengrass`套件。
- 具有不含任何引數的建構函數。
- 實作 `DeviceIdentityInterface` 界面。如需詳細資訊，請參閱[實作 `DeviceIdentityInterface` 介面](#)。

實作 DeviceIdentityInterface 介面

若要在自訂外掛程式中使用

`com.aws.greengrass.provisioning.DeviceIdentityInterface` 介面，請將 Greengrass 核新增為專案的相依性。

在自訂佈建外掛程式專案中使用 `DeviceIdentityInterface`

- 您可以新增 Greengrass 核 JAR 檔案做為程式庫，或新增 Greengrass 核做為 Maven 相依性。執行以下任意一項：
 - 若要將 Greengrass 核 JAR 檔案新增為程式庫，請下載包含 Greengrass 核 JAR AWS IoT Greengrass 的核心軟體。您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：
 - <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

您可以在 ZIP 檔案的 `lib` 資料夾中找到 Greengrass 核 JAR 檔案 (`Greengrass.jar`)。將此 JAR 檔案新增至您的專案。

- 若要在 Maven 專案中使用 Greengrass 核，請在 `com.aws.greengrass` 群組中的 `nucleus` 成品上新增相依性。您也必須新增 `greengrass-common` 儲存庫，因為 Maven Central Repository 中無法使用 Greengrass 核。

```
<project ...>
  ...
  <repositories>
    <repository>
      <id>greengrass-common</id>
      <name>greengrass common</name>
      <url>https://d2jrmugq4soidf.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>
```

DeviceIdentityInterface 介面

`com.aws.greengrass.provisioning.DeviceIdentityInterface` 介面的形狀如下。

Note

您也可以 [在 GitHub 上](#) [Greengrass 核原始碼](#) 的 [com.aws.greengrass.provisioning 套件](#) 中探索這些類別。

```
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 {}
```

SystemConfiguration 和 中的每個組態值NucleusConfiguration都需要安裝 AWS IoT Greengrass Core 軟體，但您可以傳回 null。如果您的自訂佈建外掛程式傳回null任何組態值，則必須在建立 config.yaml 檔案以提供給 AWS IoT Greengrass Core 軟體安裝程式時，在系統或核心組態中提供該值。如果您的自訂佈建外掛程式針對您也在 中定義的選項傳回非空值config.yaml，則安裝程式會將 中的值取代config.yaml為外掛程式傳回的值。

安裝程式引數

AWS IoT Greengrass 核心軟體包含 安裝程式，可設定軟體並佈建 Greengrass 核心裝置執行所需的 AWS 資源。安裝程式包含下列引數，您可以指定這些引數來設定安裝：

-h, --help

(選用) 顯示安裝程式的說明資訊。

--version

(選用) 顯示 AWS IoT Greengrass 核心軟體版本。

-Droot

(選用) 要用作 AWS IoT Greengrass Core 軟體根資料夾的路徑。

Note

此引數會設定 JVM 屬性，因此您必須在執行安裝程式-jar之前指定它。例如，指定 java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar。

預設：

- Linux：~/.greengrass
- Windows：%USERPROFILE%/.greengrass

-ar, --aws-region

Core AWS IoT Greengrass 軟體用來擷取或建立其必要 AWS 資源 AWS 區域 的。

`-p, --provision`

(選用) 您可以將此裝置註冊為 AWS IoT 物件，AWS 並佈建核心裝置所需的資源。如果您指定 `true`，AWS IoT Greengrass Core 軟體會佈建 AWS IoT 物件、(選用) AWS IoT 物件群組、IAM 角色和 AWS IoT 角色別名。

預設：`false`

`-tn, --thing-name`

(選用) 您註冊為此核心裝置的 AWS IoT 物件名稱。如果名為 的物件不存在於您的 中 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立它。

 Note

物件名稱不能包含冒號 (:) 字元。

您必須指定 `--provision true` 才能套用此引數。

預設：`GreengrassV2IotThing_` 加上隨機 UUID。

`-tgn, --thing-group-name`

(選用) 您新增此核心裝置 AWS IoT AWS IoT 物件的物件群組名稱。如果部署以此物件群組為目標，則此核心裝置會在連線到 時接收該部署 AWS IoT Greengrass。如果您的 中不存在具有此名稱的物件群組 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立它。

 Note

物件群組名稱不能包含冒號 (:) 字元。

您必須指定 `--provision true` 才能套用此引數。

`-tpn, --thing-policy-name`

此功能適用於 [Greengrass 核元件](#) 的 v2.4.0 和更新版本。

(選用) 要連接到此核心裝置 AWS IoT 物件憑證 AWS IoT 的政策名稱。如果您的 中不存在具有此名稱 AWS IoT 的政策 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立該政策。

AWS IoT Greengrass 核心軟體預設會建立寬鬆 AWS IoT 政策。您可以縮小此政策的範圍，或建立自訂政策，以限制使用案例的許可。如需詳細資訊，請參閱[AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。

您必須指定 `--provision true` 才能套用此引數。

預設：GreengrassV2IoTThingPolicy

`-trn, --tes-role-name`

(選用) 用來取得 AWS 憑證的 IAM 角色名稱，讓核心裝置與服務互動 AWS。如果您的中不存在具有此名稱的角色 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會使用 GreengrassV2TokenExchangeRoleAccess 政策建立該角色。此角色無法存取您託管元件成品的 S3 儲存貯體。因此，您必須在建立元件時，將許可新增至成品的 S3 儲存貯體和物件。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

您必須指定 `--provision true` 才能套用此引數。

預設：GreengrassV2TokenExchangeRole

`-tra, --tes-role-alias-name`

(選用) 指向提供此核心裝置 AWS 登入資料的 IAM 角色 AWS IoT 之角色別名的名稱。如果您的中不存在具有此名稱的角色別名 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立該別名，並將其指向您指定的 IAM 角色。

您必須指定 `--provision true` 才能套用此引數。

預設：GreengrassV2TokenExchangeRoleAlias

`-ss, --setup-system-service`

(選用) 您可以將 AWS IoT Greengrass Core 軟體設定為在此裝置開機時執行的系統服務。系統服務名稱為 greengrass。如需詳細資訊，請參閱[將 Greengrass 核設定為系統服務](#)。

在 Linux 作業系統上，此引數要求系統化初始化系統可在裝置上使用。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

預設：false

`-u, --component-default-user`

Core AWS IoT Greengrass 軟體用來執行元件的使用者名稱或 ID。例如，您可以指定 **ggc_user**。當您在 Windows 作業系統上執行安裝程式時，需要此值。

在 Linux 作業系統上，您也可以選擇性地指定 群組。指定以冒號分隔的使用者和群組。例如：**ggc_user:ggc_group**。

下列其他考量事項適用於 Linux 作業系統：

- 如果您以根身分執行，預設元件使用者即為組態檔案中定義的使用者。如果組態檔案未定義使用者，則預設為 ggc_user:ggc_group。如果 ggc_user 或 ggc_group 不存在，軟體會建立它們。
- 如果您以非根使用者身分執行，則 AWS IoT Greengrass Core 軟體會使用該使用者來執行元件。
- 如果您未指定群組，AWS IoT Greengrass 則 Core 軟體會使用系統使用者的主要群組。

如需詳細資訊，請參閱[設定執行元件的使用者](#)。

`-d, --deploy-dev-tools`

(選用) 您可以將 [Greengrass CLI](#) 元件下載並部署到此核心裝置。您可以使用此工具來開發和偵錯此核心裝置上的元件。

Important

我們建議您僅在開發環境中使用此元件，而不是在生產環境中。此元件可讓您存取生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

您必須指定 `--provision true` 才能套用此引數。

預設：`false`

`-init, --init-config`

(選用) 用於安裝 AWS IoT Greengrass Core 軟體的組態檔案路徑。您可以使用此選項來設定具有特定核組態的新核心裝置，例如。

 Important

您指定的組態檔案會與核心裝置上的現有組態檔案合併。這包括核心裝置上的元件和元件組態。我們建議組態檔案只列出您嘗試變更的組態。

`-tp, --trusted-plugin`

(選用) 要載入為信任外掛程式的 JAR 檔案路徑。使用此選項提供佈建外掛程式 JAR 檔案，例如使用[機群佈建](#)或[自訂佈建](#)安裝，或使用私有金鑰和憑證在[硬體安全模組](#)中安裝。

`-s, --start`

(選用) 您可以在 AWS IoT Greengrass 核心軟體安裝後啟動，並選擇性地佈建資源。

預設：`true`

執行 AWS IoT Greengrass 核心軟體

安裝 [AWS IoT Greengrass Core 軟體](#) 之後，請執行它以連接您的裝置 AWS IoT Greengrass。

安裝 AWS IoT Greengrass Core 軟體時，您可以指定是否要將其安裝為具有 [systemd](#) 的系統服務。如果您選擇此選項，安裝程式會為您執行軟體，並將其設定為在裝置開機時執行。

 Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

主題

- [檢查 AWS IoT Greengrass Core 軟體是否以系統服務的形式執行](#)
- [執行 AWS IoT Greengrass Core 軟體做為系統服務](#)
- [在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體](#)

檢查 AWS IoT Greengrass Core 軟體是否以系統服務的形式執行

安裝 AWS IoT Greengrass Core 軟體時，您可以指定 `--setup-system-service true` 引數，將 AWS IoT Greengrass Core 軟體安裝為系統服務。Linux 裝置需要[系統化](#)初始化系統，才能將 AWS

IoT Greengrass Core 軟體設定為系統服務。如果您使用此選項，安裝程式會為您執行軟體，並將其設定為在裝置開機時執行。如果安裝程式成功將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會輸出下列訊息。

```
Successfully set up Nucleus as a system service
```

如果您之前已安裝 AWS IoT Greengrass Core 軟體，但沒有安裝程式輸出，您可以檢查軟體是否安裝為系統服務。

檢查 AWS IoT Greengrass Core 軟體是否已安裝為系統服務

- 執行下列命令來檢查 Greengrass 系統服務的狀態。

Linux or Unix (systemd)

```
sudo systemctl status greengrass.service
```

如果 AWS IoT Greengrass Core 軟體安裝為系統服務且處於作用中狀態，則回應看起來與下列範例類似。

```
# 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...
```

如果找不到 `systemctl` 或 `greengrass.service`，則 AWS IoT Greengrass Core 軟體不會安裝為系統服務。若要執行軟體，請參閱 [在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體](#)。

Windows Command Prompt (CMD)

```
sc query greengrass
```

如果 AWS IoT Greengrass Core 軟體安裝為 Windows 服務且處於作用中狀態，則回應看起來與下列範例類似。

```
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
```

如果 AWS IoT Greengrass Core 軟體安裝為 Windows 服務且處於作用中狀態，則回應看起來與下列範例類似。

Status	Name	DisplayName
Running	greengrass	greengrass

執行 AWS IoT Greengrass Core 軟體做為系統服務

如果 AWS IoT Greengrass Core 軟體已安裝為系統服務，您可以使用系統服務管理員來啟動、停止和管理軟體。如需詳細資訊，請參閱[將 Greengrass 核設定為系統服務](#)。

執行 AWS IoT Greengrass 核心軟體

- 執行下列命令來啟動 AWS IoT Greengrass Core 軟體。

Linux or Unix (systemd)

```
sudo systemctl start greengrass.service
```

Windows Command Prompt (CMD)

```
sc start greengrass
```


PowerShell

```
Start-Service greengrass
```

在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體

在 Linux 核心裝置上，如果 AWS IoT Greengrass Core 軟體未安裝為系統服務，您可以執行軟體的載入器指令碼來執行軟體。

在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體

- 執行下列命令來啟動 AWS IoT Greengrass Core 軟體。如果您在終端機中執行此命令，則必須保持終端機工作階段開啟，以保持 AWS IoT Greengrass 核心軟體的執行。
- 以您使用的 Greengrass 根資料夾取代 `/greengrass/v2` 或 `C#\greengrass\v2`。

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

如果成功啟動，軟體會列印下列訊息。

```
Launched Nucleus successfully.
```

在 Docker 容器中執行 AWS IoT Greengrass 核心軟體

AWS IoT Greengrass 可以設定為在 Docker 容器中執行。Docker 是一種平台，可為您提供工具來建置、執行、測試和部署以 Linux 容器為基礎的應用程式。當您執行 AWS IoT Greengrass Docker 映像時，您可以選擇是否要將 AWS 登入資料提供給 Docker 容器，並允許 AWS IoT Greengrass Core 軟體安裝程式自動佈建 AWS Greengrass 核心裝置操作所需的資源。如果您不想提供 AWS 登入資料，則可以手動佈建 AWS 資源，並在 Docker 容器中執行 AWS IoT Greengrass Core 軟體。

主題

- [支援平台和需求](#)
- [AWS IoT Greengrass Docker 軟體下載](#)
- [選擇如何佈建 AWS 資源](#)
- [從 Dockerfile 建置 AWS IoT Greengrass 容器映像](#)

- [AWS IoT Greengrass 在具有自動資源佈建的 Docker 容器中執行](#)
- [AWS IoT Greengrass 在具有手動資源佈建的 Docker 容器中執行](#)
- [在 Docker AWS IoT Greengrass 容器中進行故障診斷](#)

支援平台和需求

主機電腦必須符合下列最低需求，才能在 Docker 容器中安裝和執行 AWS IoT Greengrass Core 軟體：

- 具有網際網路連線的 Linux 作業系統。
- [Docker 引擎](#) 18.09 版或更新版本。
- (選用) [Docker Compose](#) 1.22 版或更新版本。只有在您想要使用 Docker Compose CLI 執行 Docker 映像時，才需要 Docker Compose。

若要在 Docker 容器內執行 Lambda 函數元件，您必須設定容器以符合其他需求。如需詳細資訊，請參閱[Lambda 函數需求](#)。

在程序模式中執行元件

AWS IoT Greengrass 不支援在 Docker AWS 容器內的隔離執行時間環境中執行 Lambda AWS IoT Greengrass 函數或提供的元件。您必須在沒有任何隔離的情況下，以程序模式執行這些元件。

當您設定 Lambda 函數元件時，請將隔離模式設定為無容器。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

當您部署下列任何 AWS 提供的元件時，請更新每個元件的組態，將 containerMode 參數設定為 NoContainer。如需組態更新的詳細資訊，請參閱 [更新元件組態](#)。

- [CloudWatch 指標](#)
- [Device Defender](#)
- [Firehose](#)
- [Modbus-RTU 通訊協定轉接器](#)
- [Amazon SNS](#)

AWS IoT Greengrass Docker 軟體下載

AWS IoT Greengrass 提供 Dockerfile 來建置容器映像，其具有安裝在 Amazon Linux 2 (x86_64) 基礎映像上的 AWS IoT Greengrass 核心軟體和相依性。您可以修改 Dockerfile 中的基礎映像，以在不同的平台架構 AWS IoT Greengrass 上執行。

從 [GitHub](#) 下載 Dockerfile 套件。

Dockerfile 使用較舊版本的 Greengrass。您應該更新 檔案，以使用您想要的 Greengrass 版本。如需從 Dockerfile 建置 AWS IoT Greengrass 容器映像的相關資訊，請參閱 [從 Dockerfile 建置 AWS IoT Greengrass 容器映像](#)。

選擇如何佈建 AWS 資源

當您在 Docker 容器中安裝 AWS IoT Greengrass Core 軟體時，您可以選擇是否要自動佈建 Greengrass 核心裝置操作所需的 AWS 資源，還是使用您手動佈建的資源。

- 自動資源佈建 - 當您第一次執行 AWS IoT Greengrass 容器映像時，安裝程式會佈建 AWS IoT 物件、AWS IoT 物件群組、IAM 角色和 AWS IoT 角色別名。安裝程式也可以將本機開發工具部署到核心裝置，因此您可以使用裝置來開發和測試自訂軟體元件。若要自動佈建這些資源，您必須提供 AWS 登入資料做為 Docker 映像的環境變數。

若要使用自動佈建，您必須設定 Docker 環境變數 `PROVISION=true` 並掛載登入資料檔案，將登入資料提供給 AWS 容器。

- 手動資源佈建 - 如果您不想提供 AWS 憑證給容器，則可以在 AWS 執行 AWS IoT Greengrass 容器映像之前手動佈建資源。您必須建立組態檔案，將這些資源的相關資訊提供給 Docker 容器中 AWS IoT Greengrass 的核心軟體安裝程式。

若要使用手動佈建，您必須設定 Docker 環境變數 `PROVISION=false`。手動佈建是預設選項。

如需詳細資訊，請參閱 [從 Dockerfile 建置 AWS IoT Greengrass 容器映像](#)。

從 Dockerfile 建置 AWS IoT Greengrass 容器映像

AWS 提供 Dockerfile，您可以下載並使用它在 Docker 容器中執行 AWS IoT Greengrass Core 軟體。Dockerfiles 包含用於建置 AWS IoT Greengrass 容器映像的原始碼。

在建置 AWS IoT Greengrass 容器映像之前，您必須設定 Dockerfile 以選取您要安裝 AWS IoT Greengrass 的核心軟體版本。您也可以設定環境變數，以選擇如何在安裝期間佈建資源，並自訂其他安裝選項。本節說明如何從 AWS IoT Greengrass Dockerfile 設定和建置 Docker 映像。

下載 Dockerfile 套件

您可以從 GitHub 下載 AWS IoT Greengrass Dockerfile 套件：

[AWS Greengrass Docker 儲存庫](#)

下載套件之後，請將內容解壓縮到電腦上的 *download-directory/aws-greengrass-docker-nucleus-version* 資料夾。Dockerfile 使用較舊版本的 Greengrass。您應該更新 檔案，以使用您想要的 Greengrass 版本。

指定 AWS IoT Greengrass 核心軟體版本

在 Dockerfile 中使用下列建置引數，指定您要在 Docker AWS IoT Greengrass 映像中使用的 AWS IoT Greengrass Core 軟體版本。根據預設，Dockerfile 會使用最新版本的 AWS IoT Greengrass Core 軟體。

GREENGRASS_RELEASE_VERSION

Core AWS IoT Greengrass 軟體的版本。根據預設，Dockerfile 會下載 Greengrass 核的最新可用版本。將 值設定為您要下載的 nucleus 版本。

設定環境變數

環境變數可讓您自訂 AWS IoT Greengrass Core 軟體在 Docker 容器中的安裝方式。您可以透過各種方式設定 Docker AWS IoT Greengrass 映像的環境變數。

- 若要使用相同的環境變數來建立多個映像，請直接在 Dockerfile 中設定環境變數。
- 如果您使用 `docker run` 啟動容器，請在 命令中將環境變數做為引數傳遞，或在環境變數檔案中設定環境變數，然後將檔案做為引數傳遞。如需在 Docker 中設定環境變數的詳細資訊，請參閱 Docker 文件中的[環境變數](#)。
- 如果您使用 `docker-compose up` 啟動容器，請在環境變數檔案中設定環境變數，然後將檔案做為引數傳遞。如需在 Compose 中設定環境變數的詳細資訊，請參閱 [Docker 文件](#)。

您可以為 Docker AWS IoT Greengrass 映像設定下列環境變數。

Note

請勿修改 Dockerfile 中的 TINI_KILL_PROCESS_GROUP 變數。此變數允許轉送 SIGTERM 至 PIDs 群組中的所有 PID，以便在 Docker 容器停止時 AWS IoT Greengrass，核心軟體可以正確關閉。

GGC_ROOT_PATH

(選用) 容器內資料夾的路徑，做為 AWS IoT Greengrass Core 軟體的根目錄。

預設：/greengrass/v2

PROVISION

(選用) 判斷 AWS IoT Greengrass 核心是否佈建 AWS 資源。

- 如果您指定 true，AWS IoT Greengrass Core 軟體會將容器映像註冊為 AWS IoT 物件，並佈建 Greengrass 核心裝置所需的 AWS 資源。Core AWS IoT Greengrass 軟體會佈建 AWS IoT 物件、(選用) AWS IoT 物件群組、IAM 角色和 AWS IoT 角色別名。如需詳細資訊，請參閱 [AWS IoT Greengrass 在具有自動資源佈建的 Docker 容器中執行](#)。
- 如果您指定 false，則必須建立組態檔案，以提供給 AWS IoT Greengrass 核心安裝程式，指定使用您手動建立 AWS 的資源和憑證。如需詳細資訊，請參閱 [AWS IoT Greengrass 在具有手動資源佈建的 Docker 容器中執行](#)。

預設：false

AWS_REGION

(選用) AWS IoT Greengrass 核心軟體用來擷取或建立必要 AWS 資源 AWS 區域的。

預設：us-east-1。

THING_NAME

(選用) 您註冊為此核心裝置的 AWS IoT 物件名稱。如果您的中不存在具有此名稱的物件 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立它。

您必須指定 PROVISION=true 才能套用此引數。

預設：GreengrassV2IotThing_加上隨機 UUID。

THING_GROUP_NAME

(選用) 您新增此核心裝置的 AWS IoT 物件群組名稱。如果部署以此物件群組為目標，則該群組中的此和其他核心裝置會在其連線時接收該部署 AWS IoT Greengrass。如果您的中不存在具有此名稱的物件群組 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立它。

您必須指定 `PROVISION=true` 才能套用此引數。

TES_ROLE_NAME

(選用) IAM 角色的名稱，用來取得可讓 Greengrass 核心裝置與 AWS 服務互動的 AWS 登入資料。如果您的中不存在具有此名稱的角色 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會使用 `GreengrassV2TokenExchangeRoleAccess` 政策建立該角色。此角色無法存取託管元件成品的 S3 儲存貯體。因此，您必須在建立元件時，將許可新增至成品的 S3 儲存貯體和物件。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

預設：GreengrassV2TokenExchangeRole

TES_ROLE_ALIAS_NAME

(選用) 指向為 Greengrass 核心裝置提供 AWS 登入資料之 IAM 角色 AWS IoT 的角色別名名稱。如果您的中不存在具有此名稱的角色別名 AWS 帳戶，則 AWS IoT Greengrass Core 軟體會建立它，並將其指向您指定的 IAM 角色。

預設：GreengrassV2TokenExchangeRoleAlias

COMPONENT_DEFAULT_USER

(選用) AWS IoT Greengrass 核心軟體用來執行元件之系統使用者和群組的名稱或 ID。指定使用者和群組，以冒號分隔。群組為選用項目。例如，您可以指定 **ggc_user:ggc_group** 或 **ggc_user**。

- 如果您以根身分執行，這會預設為組態檔案定義的使用者和群組。如果組態檔案未定義使用者和群組，則預設為 `ggc_user:ggc_group`。如果 `ggc_user` 或 `ggc_group` 不存在，軟體會建立它們。
- 如果您以非根使用者身分執行，AWS IoT Greengrass Core 軟體會使用該使用者來執行元件。
- 如果您未指定群組，則 AWS IoT Greengrass Core 軟體會使用系統使用者的主要群組。

如需詳細資訊，請參閱[設定執行元件的使用者](#)。

DEPLOY_DEV_TOOLS

定義是否要在容器映像中下載和部署 [Greengrass CLI 元件](#)。您可以使用 Greengrass CLI 在本機開發和偵錯元件。

 Important

建議您僅在開發環境中使用此元件，而非生產環境中。此元件可讓您存取生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

預設：false

INIT_CONFIG

(選用) 用於安裝 AWS IoT Greengrass Core 軟體的組態檔案路徑。您可以使用此選項來設定具有特定核心組態的新 Greengrass 核心裝置，或指定手動佈建的資源，例如 [Amazon S3](#)。您必須將組態檔案掛載至您在此引數中指定的路徑。

TRUSTED_PLUGIN

此功能適用於 [Greengrass 核心元件](#) 的 v2.4.0 和更新版本。

(選用) 要載入為信任外掛程式的 JAR 檔案路徑。使用此選項提供佈建外掛程式 JAR 檔案，例如使用 [機群佈建](#) 或 [自訂佈建](#) 安裝。

THING_POLICY_NAME

此功能適用於 [Greengrass 核心元件](#) 的 v2.4.0 和更新版本。

(選用) 要連接到此核心裝置 AWS IoT 物件憑證 AWS IoT 的政策名稱。如果具有此名稱 AWS IoT 的政策不存在 AWS 帳戶 於您的 AWS IoT Greengrass Core 軟體中。

您必須指定 PROVISION=true 才能套用此引數。

 Note

Core AWS IoT Greengrass 軟體預設會建立寬鬆 AWS IoT 政策。您可以縮小此政策的範圍，或建立自訂政策，以限制使用案例的許可。如需詳細資訊，請參閱 [AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。

指定要安裝的相依性

Dockerfile 中的 RUN AWS IoT Greengrass 指令會準備容器環境來執行 AWS IoT Greengrass Core 軟體安裝程式。您可以自訂 AWS IoT Greengrass Core 軟體安裝程式在 Docker 容器中執行之前安裝的相依性。

建置 AWS IoT Greengrass 映像

使用 AWS IoT Greengrass Dockerfile 建置 AWS IoT Greengrass 容器映像。您可以使用 Docker CLI 或 Docker Compose CLI 來建置映像並啟動容器。您也可以使用 Docker CLI 建置映像，然後使用 Docker Compose 從該映像啟動容器。

Docker

1. 在主機電腦上，執行下列命令來切換到包含已設定 Dockerfile 的目錄。

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. 執行下列命令，從 Dockerfile 建置 AWS IoT Greengrass 容器映像。

```
sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" .
```

Docker Compose

1. 在主機電腦上，執行下列命令來切換到包含 Dockerfile 和 Compose 檔案的目錄。

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. 執行下列命令，使用 Compose 檔案來建置 AWS IoT Greengrass 容器映像。

```
docker-compose -f docker-compose.yml build
```

您已成功建立 AWS IoT Greengrass 容器映像。Docker 映像檔已安裝 AWS IoT Greengrass 核心軟體。您現在可以在 Docker 容器中執行 AWS IoT Greengrass Core 軟體。

AWS IoT Greengrass 在具有自動資源佈建的 Docker 容器中執行

本教學課程說明如何使用自動佈建 AWS 的資源和本機開發工具，在 Docker 容器中安裝和執行 AWS IoT Greengrass Core 軟體。您可以使用此開發環境來探索 Docker 容器中 AWS IoT Greengrass 的功能。軟體需要 AWS 登入資料才能佈建這些資源並部署本機開發工具。

如果您無法提供 AWS 登入資料給容器，您可以佈建 AWS 核心裝置操作所需的資源。您也可以將開發工具部署到核心裝置，以用作開發裝置。這可讓您在執行容器時為裝置提供較少的許可。如需詳細資訊，請參閱[AWS IoT Greengrass 在具有手動資源佈建的 Docker 容器中執行](#)。

先決條件

若要完成本教學課程，您需要下列項目。

- AWS 帳戶。如果您沒有帳戶，請參閱 [設定 AWS 帳戶](#)。
- 具有為 Greengrass AWS 核心裝置佈建 AWS IoT 和 IAM 資源許可的 IAM 使用者。Core AWS IoT Greengrass 軟體安裝程式使用您的 AWS 登入資料來自動佈建這些資源。如需自動佈建資源的最低 IAM 政策資訊，請參閱 [安裝程式佈建資源的最低 IAM 政策](#)。
- Docker AWS IoT Greengrass 映像。您可以從 [Dockerfile AWS IoT Greengrass 建置映像](#)。
- 您執行 Docker 容器的主機電腦必須符合下列要求：
 - 具有網際網路連線的 Linux 作業系統。
 - [Docker 引擎](#) 18.09 版或更新版本。
 - (選用) [Docker Compose](#) 1.22 版或更新版本。只有在您想要使用 Docker Compose CLI 執行 Docker 映像時，才需要 Docker Compose。

設定您的 AWS 登入資料

在此步驟中，您會在包含安全 AWS 登入資料的主機電腦上建立登入資料檔案。當您執行 AWS IoT Greengrass Docker 映像時，您必須在 Docker 容器中將包含此登入資料檔案的資料夾掛載至 `/root/.aws/`。AWS IoT Greengrass 安裝程式使用這些登入資料來佈建中的資源 AWS 帳戶。如需安裝程式自動佈建資源所需的最低 IAM 政策的相關資訊，請參閱 [安裝程式佈建資源的最低 IAM 政策](#)。

1. 擷取下列其中一項。

- IAM 使用者的長期登入資料。如需如何擷取長期憑證的資訊，請參閱《[IAM 使用者指南](#)》中的[管理 IAM 使用者的存取金鑰](#)。
- (建議) IAM 角色的暫時登入資料。如需如何擷取臨時登入資料的資訊，請參閱《IAM 使用者指南》中的[搭配使用臨時安全登入 AWS CLI](#)資料。

2. 建立放置登入資料檔案的資料夾。

```
mkdir ./greengrass-v2-credentials
```

3. 使用文字編輯器在 `./greengrass-v2-credentials` 資料夾中建立名為 `credentials` 的組態檔案。

例如，您可以執行下列命令，使用 GNU nano 建立 `credentials` 檔案。

```
nano ./greengrass-v2-credentials/credentials
```

- 以下列格式將您的 AWS 登入資料新增至 `credentials` 檔案。

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
aws_session_token
= AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

僅包含 `aws_session_token` 暫時登入資料。

Important

啟動 AWS IoT Greengrass 容器後，請從主機電腦移除登入資料檔案。如果您不移除登入資料檔案，您的 AWS 登入資料將保持掛載在容器內。如需詳細資訊，請參閱[在容器中執行 AWS IoT Greengrass 核心軟體](#)。

建立環境檔案

本教學課程使用環境檔案來設定將傳遞至 Docker 容器內 AWS IoT Greengrass Core 軟體安裝程式的環境變數。您也可以使用 `docker run` 命令中使用 `-e` 或 `--env` 引數，在 Docker 容器中設定環境變數，也可以在 `docker-compose.yml` 檔案的 [environment 區塊](#) 中設定變數。

- 使用文字編輯器來建立名為 `.env` 的環境檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令，使用 GNU nano `.env` 在目前的目錄中建立。

```
nano .env
```

- 將下列內容複製到 檔案。

```
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
```

然後，取代下列值。

- */greengrass/v2*。您要用於安裝的 Greengrass 根資料夾。您可以使用 GGC_ROOT 環境變數來設定此值。
- *##*。您建立資源 AWS 區域的。
- *MyGreengrassCore*。物件的名稱 AWS IoT。如果物件不存在，安裝程式會建立它。安裝程式會下載憑證以驗證為 AWS IoT 實物。
- *MyGreengrassCoreGroup*。AWS IoT 物件群組的名稱。如果物件群組不存在，安裝程式會建立它，並將該物件新增至其中。如果物件群組存在且具有作用中部署，核心裝置會下載並執行部署指定的軟體。
- *GreengrassV2TokenExchangeRole*。以允許 Greengrass 核心裝置取得臨時 AWS 登入資料的 IAM 字符交換角色名稱取代。如果角色不存在，安裝程式會建立它，並建立和連接名為 *GreengrassV2TokenExchangeRole* Access 的政策。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。
- *GreengrassCoreTokenExchangeRoleAlias*。字符交換角色別名。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM 字符交換角色。如需詳細資訊，請參閱

Note

您可以將 DEPLOY_DEV_TOOLS 環境變數設定為 true 以部署 [Greengrass CLI 元件](#)，這可讓您在 Docker 容器內開發自訂元件。建議您僅在開發環境中使用此元件，而非生產環境中。此元件可讓您存取生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

在容器中執行 AWS IoT Greengrass 核心軟體

本教學課程說明如何啟動您在 Docker 容器中建置的 Docker 映像。您可以使用 Docker CLI 或 Docker Compose CLI 在 Docker 容器中執行 AWS IoT Greengrass 核心軟體映像。

Docker

1. 執行下列命令來啟動 Docker 容器。

```
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
```

此範例命令使用下列引數來執行 [docker](#)：

- [--rm](#)。在容器結束時清除容器。
- [--init](#)。在容器中使用初始化程序。

 Note

當您停止 Docker 容器時，需要 `--init` 引數來關閉 AWS IoT Greengrass Core 軟體。

- [-it](#)。（選用）在前景中以互動式程序執行 Docker 容器。您可以使用 `-d` 引數取代此值，以分離模式執行 Docker 容器。如需詳細資訊，請參閱 Docker 文件中的 [分離與前景](#)。
- [--name](#)。執行名為 `aws-iot-greengrass` 的容器。
- [-v](#)。將磁碟區掛載到 Docker 容器，讓組態檔案和憑證檔案可供容器內 AWS IoT Greengrass 執行。
- [--env-file](#)。（選用）指定環境檔案，以設定要傳遞至 Docker 容器內 AWS IoT Greengrass Core 軟體安裝程式的環境變數。只有在您建立 [環境檔案](#) 來設定環境變數時，才需要此引數。如果您未建立環境檔案，您可以使用 `--env` 引數直接在 Docker 執行命令中設定環境變數。
- [-p](#)。（選用）將 8883 容器連接埠發佈至主機。如果您想要透過 MQTT 連接和通訊，則需要此引數，因為 AWS IoT Greengrass 使用連接埠 8883 處理 MQTT 流量。若要開啟其他連接埠，請使用其他 `-p` 數。

 Note

若要以更高的安全性執行 Docker 容器，您可以使用 `--cap-drop` 和 `--cap-add` 引數來選擇性地為您的容器啟用 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的 [執行期權限和 Linux 功能](#)。

2. 在主機裝置 `./greengrass-v2-credentials` 上從 移除登入資料。

```
rm -rf ./greengrass-v2-credentials
```

Important

您正在移除這些登入資料，因為它們提供核心裝置只在設定期間需要的廣泛許可。如果您未移除這些登入資料，Greengrass 元件和容器中執行的其他程序可以存取這些登入資料。如果您需要提供 AWS 憑證給 Greengrass 元件，請使用字符交換服務。如需詳細資訊，請參閱[與 AWS 服務互動](#)。

Docker Compose

1. 使用文字編輯器來建立名為 `docker-compose.yml` 的 Docker Compose 檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令，使用 GNU nano `docker-compose.yml` 在目前的目錄中建立。

```
nano docker-compose.yml
```

Note

您也可以從 [GitHub](#) 下載並使用 AWS 最新版本的 Compose 檔案。

2. 將下列內容新增至 Compose 檔案。您的檔案看起來應該如下列範例：將 *docker-image* 取代為 Docker 映像的名稱。

```
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"
```

此範例中的下列參數 Compose 檔案為選用：

- `ports`—發佈 8883 容器連接埠至主機。如果您想要透過 MQTT 連接和通訊，則需要此參數，因為 AWS IoT Greengrass 使用連接埠 8883 處理 MQTT 流量。
- `env_file`- 指定環境檔案，以設定將傳遞至 Docker 容器內 AWS IoT Greengrass Core 軟體安裝程式的環境變數。只有在您建立[環境檔案](#)來設定環境變數時，才需要此參數。如果您未建立環境檔案，您可以使用[環境](#)參數直接在 Compose 檔案中設定變數。

 Note

若要以更高的安全性執行 Docker 容器，您可以在 `cap_add` Compose 檔案中使用 `cap_drop` 和 `cap_add`，為您的容器選擇性地啟用 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的[執行期權限](#)和 [Linux 功能](#)。

3. 執行下列命令來啟動 Docker 容器。

```
docker-compose -f docker-compose.yml up
```

4. 在主機裝置 `./greengrass-v2-credentials` 上從 移除登入資料。

```
rm -rf ./greengrass-v2-credentials
```

 Important

您正在移除這些登入資料，因為它們提供核心裝置只在設定期間需要的廣泛許可。如果您未移除這些登入資料，Greengrass 元件和容器中執行的其他程序可以存取這些登入資料。如果您需要提供 AWS 憑證給 Greengrass 元件，請使用字符交換服務。如需詳細資訊，請參閱[與 AWS 服務互動](#)。

後續步驟

AWS IoT Greengrass 核心軟體現在正在 Docker 容器中執行。執行下列命令，以擷取目前執行中容器的容器 ID。

```
docker ps
```

然後，您可以執行下列命令來存取容器，並探索在容器內執行 AWS IoT Greengrass 的核心軟體。

```
docker exec -it container-id /bin/bash
```

如需建立簡單元件的詳細資訊，請參閱 [步驟 4：在您的裝置上開發和測試元件](#) 中的 [教學課程：入門 AWS IoT Greengrass V2](#)

Note

當您使用在 Docker 容器內 `docker exec` 執行命令時，這些命令不會記錄在 Docker 日誌中。若要在 Docker 日誌中記錄命令，請將互動式 shell 連接到 Docker 容器。如需詳細資訊，請參閱 [將互動式 shell 連接至 Docker 容器](#)。

AWS IoT Greengrass 核心日誌檔案稱為 `greengrass.log`，位於 `/greengrass/v2/logs`。元件日誌檔案也位於相同的目錄中。若要將 Greengrass 日誌複製到主機上的暫存目錄，請執行下列命令：

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

如果您想要在容器結束或移除後保留日誌，建議您僅將 `/greengrass/v2/logs` 目錄繫結至主機上的暫存日誌目錄，而不是掛載整個 Greengrass 目錄。如需詳細資訊，請參閱 [在 Docker 容器外保留 Greengrass 日誌](#)。

若要停止執行中的 AWS IoT Greengrass Docker 容器，請執行 `docker stop` 或 `docker-compose -f docker-compose.yml stop`。此動作會 SIGTERM 傳送至 Greengrass 程序，並關閉在容器中啟動的所有相關程序。Docker 容器會使用 `docker-init` 可執行檔初始化為程序 PID 1，這有助於移除任何剩餘的殭屍程序。如需詳細資訊，請參閱 Docker 文件中的 [指定初始化程序](#)。

如需對 AWS IoT Greengrass 在 Docker 容器中執行 的問題進行故障診斷的資訊，請參閱 [在 Docker AWS IoT Greengrass 容器中進行故障診斷](#)。

AWS IoT Greengrass 在具有手動資源佈建的 Docker 容器中執行

本教學課程說明如何使用手動佈建 AWS 的資源，在 Docker 容器中安裝和執行 AWS IoT Greengrass 核心軟體。

主題

- [先決條件](#)
- [擷取 AWS IoT 端點](#)
- [建立 AWS IoT 物件](#)
- [建立物件憑證](#)
- [建立權杖交換角色](#)
- [將憑證下載至裝置](#)
- [建立組態檔案](#)
- [建立環境檔案](#)
- [在容器中執行 AWS IoT Greengrass 核心軟體](#)
- [後續步驟](#)

先決條件

為了完成本教學，您需要以下項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 [設定 AWS 帳戶](#)。
- Docker AWS IoT Greengrass 映像。您可以從 [Dockerfile AWS IoT Greengrass 建置映像](#)。
- 您執行 Docker 容器的主機電腦必須符合下列要求：
 - 具有網際網路連線的 Linux 作業系統。
 - [Docker 引擎](#) 18.09 版或更新版本。
 - (選用) [Docker Compose](#) 1.22 版或更新版本。只有在您想要使用 Docker Compose CLI 執行 Docker 映像時，才需要 Docker Compose。

擷取 AWS IoT 端點

取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置使用這些端點來連線 AWS IoT。請執行下列操作：

1. 取得 AWS IoT 的資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果請求成功，回應看起來與下列範例類似。


```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 取得的 AWS IoT 登入資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-  
west-2.amazonaws.com"
}
```

建立 AWS IoT 物件

AWS IoT 物件代表連線至的裝置和邏輯實體 AWS IoT。Greengrass 核心裝置是 AWS IoT 實物。當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。

在本節中，您會建立代表裝置的 AWS IoT 物件。

建立 AWS IoT 物件

1. 為您的裝置建立 AWS IoT 物件。在您的開發電腦上，執行下列命令。

- 將 *MyGreengrassCore* 取代為要使用的物件名稱。此名稱也是 Greengrass 核心裝置的名稱。

Note

物件名稱不能包含冒號 (:) 字元。

```
aws iot create-thing --thing-name MyGreengrassCore
```

如果請求成功，回應看起來與下列範例類似。

```
{
```

```
"thingName": "MyGreengrassCore",  
"thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",  
"thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"  
}
```

2. (選用) 將 AWS IoT 物件新增至新的或現有的物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的機群。將軟體元件部署至裝置時，您可以將個別裝置或裝置群組設為目標。您可以將裝置新增至具有作用中 Greengrass 部署的物件群組，以將該物件群組的軟體元件部署到裝置。請執行下列操作：

- a. (選用) 建立 AWS IoT 物件群組。

- 將 *MyGreengrassCoreGroup* 取代為要建立的物件群組名稱。

 Note

物件群組名稱不能包含冒號 (:) 字元。

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "thingGroupName": "MyGreengrassCoreGroup",  
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/  
MyGreengrassCoreGroup",  
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"  
}
```

- b. 將 AWS IoT 物件新增至物件群組。

- 將 *MyGreengrassCore* 取代為您的 AWS IoT 物件名稱。
- 將 *MyGreengrassCoreGroup* 取代為物件群組的名稱。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-  
name MyGreengrassCoreGroup
```

如果請求成功，命令不會有任何輸出。

建立物件憑證

當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。此憑證可讓裝置與 AWS IoT 和 通訊 AWS IoT Greengrass。

在本節中，您可以建立和下載裝置可用來連線的憑證 AWS。

建立物件憑證

1. 建立資料夾以下載 AWS IoT 物件的憑證。

```
mkdir greengrass-v2-certs
```

2. 建立並下載 AWS IoT 物件的憑證。

```
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
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificateId":  
  "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificatePem": "-----BEGIN CERTIFICATE-----  
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w  
0BAQUFADCBiDELMAKGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ  
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAwTC0lBTsBDb25zb2x1MRIw  
EAYDVQQDEw1UZXRhbnQ2IjYwMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5  
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAKGA1UEBh  
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb  
WF6b24xFDASBgNVBAwTC0lBTsBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2IjYwMx  
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wZ8wDQYJKoZIhvcNAQEE  
BQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI  
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ  
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr  
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN  
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo  
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjStbNYiytVbZPQUQ5Yaxu2jXnimvw  
3rrszlaEXAMPLE=
```

```

-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\n
MIIBIjANBgkqhkiEXAMPEQFAA0CAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\n
MMEXAMPLEUuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\n
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTtwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUSTzecyNCx2EXAMPLEvp9mQ0UXP6plfgxwKRX2fEXAMPLEDa\n
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFR188eGdsAEXAMPLElnI9EesG\n
FQIDAQAB\n
-----END PUBLIC KEY-----\n
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\n
key omitted for security reasons\n
-----END RSA PRIVATE KEY-----\n
"
  }
}

```

儲存憑證的 Amazon Resource Name (ARN)，以供稍後用於設定憑證。

接著，設定物件憑證。如需詳細資訊，請參閱[設定物件憑證](#)。

建立權杖交換角色

Greengrass 核心裝置使用稱為字符交換角色的 IAM 服務角色來授權呼叫 AWS 服務。裝置使用 AWS IoT 登入資料提供者取得此角色的臨時 AWS 登入資料，允許裝置與 互動 AWS IoT、將日誌傳送至 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

您可以使用 AWS IoT 角色別名來設定 Greengrass 核心裝置的字符交換角色。角色別名可讓您變更裝置的字符交換角色，但讓裝置組態保持不變。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權直接呼叫 AWS 服務](#)。

在本節中，您會建立權杖交換 IAM 角色和指向角色 AWS IoT 的角色別名。如果您已經設定 Greengrass 核心裝置，您可以使用其權杖交換角色和角色別名，而不是建立新的權杖交換角色和角色別名。然後，您將裝置的 AWS IoT 物件設定為使用該角色和別名。

建立權杖交換 IAM 角色

1. 建立您的裝置可用作字符交換角色的 IAM 角色。請執行下列操作：

- a. 建立檔案，其中包含權杖交換角色所需的信任政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-trust-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 使用信任政策文件建立權杖交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為要建立的 IAM 角色名稱。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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. 建立檔案，其中包含權杖交換角色所需的存取政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

 Note

此存取政策不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須將許可新增至角色，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

如果您還沒有元件成品的 S3 儲存貯體，您可以在建立儲存貯體之後新增這些許可。

d. 從政策文件中建立 IAM 政策。

- 以要建立的 IAM 政策名稱取代 *GreengrassV2TokenExchangeRoleAccess*。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --  
policy-document file://device-role-access-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "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. 將 IAM 政策連接至字符交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為 IAM 角色的名稱。
- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-  
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，命令不會有任何輸出。

2. 建立指向字符交換 AWS IoT 角色的角色別名。

- 以要建立的角色別名名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。

- 將角色 ARN 取代為您在上一個步驟中建立之 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

若要建立角色別名，您必須具有將字符交換 IAM 角色傳遞至的許可 AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的 AWS 使用者是否具有此許可。如需詳細資訊，請參閱 AWS Identity and Access Management 《使用者指南》中的 [授予使用者將角色傳遞至 AWS 服務的許可](#)。

3. 建立並連接允許 Greengrass 核心裝置使用角色別名來擔任字符交換角色 AWS IoT 的政策。如果您之前已設定 Greengrass 核心裝置，您可以連接其角色別名 AWS IoT 政策，而不是建立新的。請執行下列操作：
 - a. （選用）建立檔案，其中包含角色別名所需的 AWS IoT 政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano greengrass-v2-iot-role-alias-policy.json
```

將下列 JSON 複製到檔案。

- 使用角色別名的 ARN 取代資源 ARN。

```
{
  "Version": "2012-10-17",
  "Statement": [
```



```
{
  "Effect": "Allow",
  "Action": "iot:AssumeRoleWithCertificate",
  "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
]
```

b. 從 AWS IoT 政策文件中建立政策。

- 將 *GreengrassCoreTokenExchangeRoleAliasPolicy* 取代為要建立 AWS IoT 的政策名稱。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "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. 將 AWS IoT 政策連接至 AWS IoT 物件的憑證。

- 以角色別名 AWS IoT 政策的名稱取代 *GreengrassCoreTokenExchangeRoleAliasPolicy*。

- 將目標 ARN 取代為物件 AWS IoT 憑證的 ARN。

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，命令不會有任何輸出。

將憑證下載至裝置

先前，您已將裝置的憑證下載到開發電腦。在本節中，您會下載 Amazon 根憑證授權機構 (CA) 憑證。然後，如果您計劃在開發電腦以外的電腦上在 Docker 中執行 AWS IoT Greengrass Core 軟體，您可以將憑證複製到該主機電腦。Core AWS IoT Greengrass 軟體使用這些憑證來連線至 AWS IoT 雲端服務。

將憑證下載至裝置

1. 在您的開發電腦上，下載 Amazon 根憑證授權機構 (CA) certificate. AWS IoT certificates 預設會與 Amazon 根憑證授權機構憑證相關聯。

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. 如果您打算在與開發電腦不同的裝置上在 Docker 中執行 AWS IoT Greengrass Core 軟體，請將憑證複製到主機電腦。如果在開發電腦和主機電腦上啟用 SSH 和 SCP，您可以使用開發電腦上的 scp 命令來傳輸憑證。將 *device-ip-address* 取代為主機電腦的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

建立組態檔案

1. 在主機電腦上，建立您放置組態檔案的資料夾。

```
mkdir ./greengrass-v2-config
```

2. 使用文字編輯器在 `./greengrass-v2-config` 資料夾中建立名為 `config.yaml` 的組態檔案。

例如，您可以執行下列命令來使用 GNU nano 來建立 `config.yaml`。

```
nano ./greengrass-v2-config/config.yaml
```

3. 將下列 YAML 內容複製到 檔案。此部分組態檔案會指定系統參數和 Greengrass 核參數。

```
---
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"
```

然後，取代下列值：

- */tmp/certs*。您在啟動容器時，將下載的憑證掛載到 Docker 容器中的目錄。
- */greengrass/v2*。您要用於安裝的 Greengrass 根資料夾。您可以使用 `GGC_ROOT` 環境變數來設定此值。
- *MyGreengrassCore*。物件的名稱 AWS IoT。

- *nucleus-version*。要安裝 AWS IoT Greengrass 的核心軟體版本。此值必須符合您下載的 Docker 映像或 Dockerfile 版本。如果您使用 latest 標籤下載 Greengrass Docker 映像，請使用 `docker inspect image-id` 查看映像版本。
- *##*。您建立 AWS IoT 資源 AWS 區域的。您也必須為 `AWS_REGION` 環境 [檔案中](#) 的環境變數指定相同的值。
- *GreengrassCoreTokenExchangeRoleAlias*。字符交換角色別名。
- *device-data-prefix*。AWS IoT 資料端點的字首。
- *device-credentials-prefix*。登入 AWS IoT 資料端點的字首。

建立環境檔案

本教學課程使用環境檔案來設定將傳遞至 Docker 容器內 AWS IoT Greengrass Core 軟體安裝程式的環境變數。您也可以使用 `docker run` 命令中使用 [-e 或 --env 引數](#)，在 Docker 容器中設定環境變數，也可以在 `docker-compose.yml` 檔案的 [environment 區塊](#) 中設定變數。

1. 使用文字編輯器來建立名為 `.env` 的環境檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令，使用 GNU nano `.env` 在目前的目錄中建立。

```
nano .env
```

2. 將下列內容複製到 檔案。

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=false
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
INIT_CONFIG=/tmp/config/config.yaml
```

然後，取代下列值。

- */greengrass/v2*。用來安裝 AWS IoT Greengrass Core 軟體的根資料夾路徑。
- *##*。您建立 AWS IoT 資源 AWS 區域的。您必須為 `awsRegion` 組態 [檔案中的組態](#) 參數指定相同的值。
- */tmp/config/*。您在啟動 Docker 容器時掛載組態檔案的資料夾。

Note

您可以將 `DEPLOY_DEV_TOOLS` 環境變數設定為 `true` 以部署 [Greengrass CLI 元件](#)，這可讓您在 Docker 容器內開發自訂元件。建議您僅在開發環境中使用此元件，而非生產環境中。此元件可讓您存取生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

在容器中執行 AWS IoT Greengrass 核心軟體

本教學課程說明如何啟動您在 Docker 容器中建置的 Docker 映像。您可以使用 Docker CLI 或 Docker Compose CLI 在 Docker 容器中執行 AWS IoT Greengrass 核心軟體映像。

Docker

- 本教學課程說明如何啟動您在 Docker 容器中建置的 Docker 映像。

```
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
```

此範例命令使用下列引數來執行 [docker](#)：

- [--rm](#)。在容器結束時清除容器。
- [--init](#)。在容器中使用初始化程序。

Note

當您停止 Docker 容器時，需要 `--init` 引數來關閉 AWS IoT Greengrass Core 軟體。

- [-it](#)。（選用）在前景中以互動式程序執行 Docker 容器。您可以使用 `-d` 引數取代此值，以分離模式執行 Docker 容器。如需詳細資訊，請參閱 Docker 文件中的 [分離與前景](#)。
- [--name](#)。執行名為 `aws-iot-greengrass` 的容器

- `-v`。將磁碟區掛載到 Docker 容器，讓組態檔案和憑證檔案可供容器內 AWS IoT Greengrass 執行。
- `--env-file`。（選用）指定環境檔案，以設定要傳遞至 Docker 容器內 AWS IoT Greengrass Core 軟體安裝程式的環境變數。只有在您建立[環境檔案](#)來設定環境變數時，才需要此引數。如果您未建立環境檔案，您可以使用 `--env` 引數直接在 Docker 執行命令中設定環境變數。
- `-p`。（選用）將 8883 容器連接埠發佈至主機。如果您想要透過 MQTT 連接和通訊，則需要此引數，因為 AWS IoT Greengrass 使用連接埠 8883 處理 MQTT 流量。若要開啟其他連接埠，請使用其他引數 `-p`。

 Note

若要以更高的安全性執行 Docker 容器，您可以使用 `--cap-drop` 和 `--cap-add` 引數來選擇性地為您的容器啟用 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的[執行期權限](#)和 [Linux 功能](#)。

Docker Compose

1. 使用文字編輯器來建立名為 `docker-compose.yml` 的 Docker Compose 檔案。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令，使用 GNU nano 在目前的目錄中建立 `docker-compose.yml`。

```
nano docker-compose.yml
```

 Note

您也可以從 [GitHub](#) 下載並使用最新版本的 AWS Compose 檔案。

2. 將下列內容新增至 Compose 檔案。您的檔案看起來應該如下列範例：將 *`your-container-name#version`* 取代為 Docker 映像的名稱。

```
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"
```

此範例中的下列參數 Compose 檔案為選用：

- `ports`—發佈 8883 容器連接埠至主機。如果您想要透過 MQTT 連接和通訊，則需要此參數，因為 AWS IoT Greengrass 使用連接埠 8883 處理 MQTT 流量。
- `env_file`—指定環境檔案，以設定將傳遞至 Docker 容器內 AWS IoT Greengrass Core 軟體安裝程式的環境變數。只有在您建立[環境檔案](#)來設定環境變數時，才需要此參數。如果您未建立環境檔案，您可以使用[環境](#)參數直接在 Compose 檔案中設定變數。

Note

若要以更高的安全性執行 Docker 容器，您可以在 `cap_add` Compose 檔案中使用 `cap_drop` 和 `cap_add`，為您的容器選擇性地啟用 Linux 功能。如需詳細資訊，請參閱 Docker 文件中的[執行期權限](#)和 [Linux 功能](#)。

3. 執行下列命令來啟動容器。

```
docker-compose -f docker-compose.yml up
```

後續步驟

AWS IoT Greengrass 核心軟體現在正在 Docker 容器中執行。執行下列命令，以擷取目前執行中容器的容器 ID。

```
docker ps
```

然後，您可以執行下列命令來存取容器，並探索在容器內執行 AWS IoT Greengrass 的核心軟體。

```
docker exec -it container-id /bin/bash
```

如需建立簡單元件的詳細資訊，請參閱 [步驟 4：在您的裝置上開發和測試元件](#) 中的 [教學課程：入門 AWS IoT Greengrass V2](#)

Note

當您使用在 Docker 容器內 `docker exec` 執行命令時，這些命令不會記錄在 Docker 日誌中。若要在 Docker 日誌中記錄命令，請將互動式 shell 連接到 Docker 容器。如需詳細資訊，請參閱 [將互動式 shell 連接至 Docker 容器](#)。

AWS IoT Greengrass 核心日誌檔案稱為 `greengrass.log`，位於 `/greengrass/v2/logs`。元件日誌檔案也位於相同的目錄中。若要將 Greengrass 日誌複製到主機上的暫存目錄，請執行下列命令：

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

如果您想要在容器結束或移除後保留日誌，建議您僅將 `/greengrass/v2/logs` 目錄繫結至主機上的暫存日誌目錄，而不是掛載整個 Greengrass 目錄。如需詳細資訊，請參閱 [在 Docker 容器外保留 Greengrass 日誌](#)。

若要停止執行中的 AWS IoT Greengrass Docker 容器，請執行 `docker stop` 或 `docker-compose -f docker-compose.yml stop`。此動作會 SIGTERM 傳送至 Greengrass 程序，並關閉在容器中啟動的所有相關程序。Docker 容器會使用 `docker-init` 可執行檔初始化為程序 PID 1，這有助於移除任何剩餘的殭屍程序。如需詳細資訊，請參閱 Docker 文件中的 [指定初始化程序](#)。

如需對 AWS IoT Greengrass 在 Docker 容器中執行 的問題進行故障診斷的資訊，請參閱 [在 Docker AWS IoT Greengrass 容器中進行故障診斷](#)。

在 Docker AWS IoT Greengrass 容器中進行故障診斷

使用下列資訊來協助您對 AWS IoT Greengrass 在 Docker 容器中執行的 問題進行故障診斷，並對 AWS IoT Greengrass Docker 容器中的 問題進行偵錯。

主題

- [對執行 Docker 容器的問題進行故障診斷](#)
- [在 Docker AWS IoT Greengrass 容器中偵錯](#)

對執行 Docker 容器的問題進行故障診斷

使用以下資訊來協助疑難排解在 Docker 容器中執行 AWS IoT Greengrass 的問題。

主題

- [錯誤：無法從非 TTY 裝置執行互動式登入](#)
- [錯誤：不明選項：-no-include-email](#)
- [錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。](#)
- [錯誤：呼叫 GetAuthorizationToken 操作時發生錯誤 \(AccessDeniedException\)：使用者：arn：aws：iam：：account-id：user/<user-name> 未獲授權在資源上執行：ecr：GetAuthorizationToken：*](#)
- [錯誤：您已經達到提取率限制](#)

錯誤：無法從非 TTY 裝置執行互動式登入

當您執行 `aws ecr get-login-password` 命令時，可能會發生此錯誤。請確定您已安裝最新版本 AWS CLI 2 或版本 1。我們建議您使用第 2 AWS CLI 版。如需詳細資訊，請參閱「AWS Command Line Interface 使用者指南」中的[安裝 AWS CLI](#)。

錯誤：不明選項：-no-include-email

當您執行 `aws ecr get-login` 命令時，可能會發生此錯誤。請確定您已安裝 AWS CLI 最新版本（例如，執行：`pip install awscli --upgrade --user`）。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的在[Microsoft Windows AWS Command Line Interface 上安裝](#)。

錯誤：防火牆封鎖了 Windows 和容器之間的檔案共用。

在 Windows 電腦上執行 Docker 時，您可能會收到此錯誤或 Firewall Detected 訊息。如果您登入虛擬私有網路 (VPN)，而您的網路設定防止掛載共用磁碟機，也可能會發生這個錯誤。在這種情況下，請關閉 VPN 並重新執行 Docker 容器。

錯誤：呼叫 GetAuthorizationToken 操作時發生錯誤 (AccessDeniedException)：使用者：arn：aws：iam：：**account-id**：user/<user-name> 未獲授權在資源上執行：ecr：GetAuthorizationToken：*

如果您沒有足夠的許可來存取 Amazon ECR 儲存庫，則執行 `aws ecr get-login-password` 命令時可能會收到此錯誤。如需詳細資訊，請參閱《[Amazon ECR 使用者指南](#)》中的[Amazon ECR 儲存庫政策範例和存取一個 Amazon ECR 儲存庫](#)。

錯誤：您已經達到提取率限制

Docker Hub 限制匿名和 Free Docker Hub 使用者可以提出的提取請求數量。如果您超過匿名或免費使用者提取請求的速率限制，則會收到下列其中一個錯誤：

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

若要解決這些錯誤，您可以等待幾個小時，然後再嘗試另一個提取請求。如果您打算持續提交大量提取請求，請參閱 [Docker Hub 網站](#)，以取得速率限制的相關資訊，以及驗證和升級 Docker 帳戶的選項。

在 Docker AWS IoT Greengrass 容器中偵錯

為了對 Docker 容器的問題進行偵錯，您可以保留 Greengrass 執行時間日誌或將互動式 shell 連接到 Docker 容器。

在 Docker 容器外保留 Greengrass 日誌

停止 AWS IoT Greengrass 容器之後，您可以使用下列 `docker cp` 命令，將 Docker 容器的 Greengrass 日誌複製到暫時日誌目錄。

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

若要即使在容器結束或移除之後仍保留日誌，您必須在繫結掛載 `/greengrass/v2/logs` 目錄之後執行 AWS IoT Greengrass Docker 容器。

若要繫結掛載 `/greengrass/v2/logs` 目錄，請在執行新的 Docker AWS IoT Greengrass 容器時執行下列其中一項操作。

- 將包含在 `docker run` 命令 `-v /tmp/logs:/greengrass/v2/logs:ro` 中。

在執行 `docker-compose up` 命令之前，修改 Compose 檔案中的 `volumes` 區塊，以包含以下行。

```
volumes:  
- /tmp/logs:/greengrass/v2/logs:ro
```

然後，您可以在主機 `/tmp/logs` 上檢查日誌，在 AWS IoT Greengrass Docker 容器內執行時查看 Greengrass 日誌。

如需有關執行 Greengrass Docker 容器的資訊，請參閱 [使用手動佈建在 Docker AWS IoT Greengrass 中執行](#) 和 [使用自動佈建在 Docker AWS IoT Greengrass 中執行](#)

將互動式 shell 連接至 Docker 容器

當您使用在 Docker 容器內 `docker exec` 執行命令時，這些命令不會擷取在 Docker 日誌中。在 Docker 日誌中記錄命令可協助您調查 Greengrass Docker 容器的狀態。執行以下任意一項：

- 在個別終端機中執行下列命令，將終端機的標準輸入、輸出和錯誤連接至執行中的容器。這可讓您從目前的終端機檢視和控制 Docker 容器。

```
docker attach container-id
```

- 在個別終端機中執行下列命令。這可讓您以互動式模式執行命令，即使未連接容器也一樣。

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

如需一般 AWS IoT Greengrass 故障診斷，請參閱 [疑難排解](#)。

設定 AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體提供可用來設定軟體的選項。您可以建立部署，在每個 AWS IoT Greengrass 核心裝置上設定 核心軟體。

主題

- [部署 Greengrass 核元件](#)
- [將 Greengrass 核設定為系統服務](#)
- [使用 JVM 選項控制記憶體配置](#)
- [設定執行元件的使用者](#)
- [設定元件的系統資源限制](#)
- [連線至連接埠 443 或透過網路代理](#)
- [使用私有 CA 簽署的裝置憑證](#)
- [設定 MQTT 逾時和快取設定](#)
- [在 IPv6 網路上設定 Greengrass Nucleus](#)

部署 Greengrass 核元件

AWS IoT Greengrass 提供 AWS IoT Greengrass Core 軟體做為元件，您可以將其部署到 Greengrass 核心裝置。您可以建立部署，將相同的組態套用至多個 Greengrass 核心裝置。如需詳細資訊，請參閱[Greengrass 核](#)及[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

將 Greengrass 核設定為系統服務

您必須將 AWS IoT Greengrass Core 軟體設定為裝置初始化系統中的系統服務，才能執行下列動作：

- 在裝置開機時啟動 AWS IoT Greengrass Core 軟體。如果您管理大型裝置機群，這是很好的做法。
- 安裝並執行外掛程式元件。數個 AWS 提供的元件是外掛程式元件，可讓它們直接與 Greengrass 核連接。如需元件類型的詳細資訊，請參閱[元件類型](#)。
- 將over-the-air(OTA) 更新套用至核心裝置 AWS IoT Greengrass 的核心軟體。如需詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。
- 當部署將元件更新至新版本或更新特定組態參數時，啟用元件以重新啟動 AWS IoT Greengrass Core 軟體或核心裝置。如需詳細資訊，請參閱[引導生命週期步驟](#)。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

主題

- [將 核設定為系統服務 \(Linux\)](#)
- [將核心設定為系統服務 \(Windows\)](#)

將 核設定為系統服務 (Linux)

Linux 裝置支援不同的初始化系統，例如 initd、Systemd 和 SystemV。當您安裝 AWS IoT Greengrass Core 軟體時，您可以使用 `--setup-system-service true` 引數，以系統服務的形式啟動核心，並將其設定為在裝置開機時啟動。安裝程式會將 AWS IoT Greengrass Core 軟體設定為具有 systemd 的系統服務。

您也可以手動設定要做為系統服務執行的 核。以下範例為 systemd 的服務檔案：

```
[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
```

設定系統服務之後，您可以執行下列命令來設定在開機時啟動裝置，以及啟動或停止 AWS IoT Greengrass Core 軟體。

- 檢查服務的狀態（系統化）

```
sudo systemctl status greengrass.service
```

- 若要在裝置開機時啟動核心。

```
sudo systemctl enable greengrass.service
```

- 在裝置開機時停止啟動核心。

```
sudo systemctl disable greengrass.service
```

- 啟動 AWS IoT Greengrass Core 軟體。

```
sudo systemctl start greengrass.service
```

- 停止 AWS IoT Greengrass Core 軟體。

```
sudo systemctl stop greengrass.service
```

將核心設定為系統服務 (Windows)

安裝 AWS IoT Greengrass 核心軟體時，您可以使用 `--setup-system-service true` 引數，以 Windows 服務的形式啟動核心，並將其設定為在裝置開機時啟動。

設定服務之後，您可以執行下列命令來設定在開機時啟動裝置，以及啟動或停止 AWS IoT Greengrass Core 軟體。您必須以管理員身分執行命令提示字元或 PowerShell，才能執行這些命令。

Windows Command Prompt (CMD)

- 檢查服務的狀態

```
sc query "greengrass"
```

- 若要在裝置開機時啟動核心。

```
sc config "greengrass" start=auto
```

- 在裝置開機時停止啟動核心。

```
sc config "greengrass" start=disabled
```

- 啟動 AWS IoT Greengrass Core 軟體。

```
sc start "greengrass"
```

- 停止 AWS IoT Greengrass Core 軟體。

```
sc stop "greengrass"
```

Note

在 Windows 裝置上，AWS IoT Greengrass Core 軟體會在關閉 Greengrass 元件程序時忽略此關閉訊號。如果 AWS IoT Greengrass Core 軟體在您執行此命令時忽略關閉訊號，請等待幾秒鐘，然後再試一次。

PowerShell

- 檢查服務的狀態

```
Get-Service -Name "greengrass"
```

- 若要在裝置開機時啟動核心。

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- 在裝置開機時停止啟動核心。

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- 啟動 AWS IoT Greengrass Core 軟體。

```
Start-Service -Name "greengrass"
```

- 停止 AWS IoT Greengrass Core 軟體。

```
Stop-Service -Name "greengrass"
```



Note

在 Windows 裝置上，AWS IoT Greengrass Core 軟體會在關閉 Greengrass 元件程序時忽略此關閉訊號。如果 AWS IoT Greengrass Core 軟體在您執行此命令時忽略關閉訊號，請等待幾秒鐘，然後再試一次。

使用 JVM 選項控制記憶體配置

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以使用 Java 虛擬機器 (JVM) 選項來控制堆積大小上限、垃圾收集模式和編譯器選項，以控制 AWS IoT Greengrass 核心軟體使用的記憶體數量。JVM 中的堆積大小會決定應用程式在[垃圾收集](#)發生之前或應用程式用盡記憶體之前可以使用多少記憶體。堆積大小的上限，指定了在繁重的作業進行期間擴充堆疊時，JVM 可以分配的最大記憶體容量。

若要控制記憶體配置，請建立新的部署或修改包含 nucleus 元件的現有部署，並在 [nucleus 元件組態](#) 中的 `jvmOptions` 組態參數中指定您的 JVM 選項。

根據您的需求，您可以執行 AWS IoT Greengrass 核心軟體，減少記憶體配置或減少記憶體配置。

減少記憶體配置

若要在記憶體配置減少的情況下執行 AWS IoT Greengrass 核心軟體，建議您使用下列範例組態合併更新，在核心組態中設定 JVM 選項：

```
{  
  "jvmOptions": "-XX:+UseSerialGC -XX:TieredStopAtLevel=1"  
}
```

記憶體配置下限

若要以最低記憶體配置執行 AWS IoT Greengrass 核心軟體，建議您使用下列範例組態合併更新，在核心組態中設定 JVM 選項：

```
{  
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"  
}
```

Important

執行具有最低記憶體配置 AWS IoT Greengrass 的核心軟體可能會對低規格系統產生重大效能影響，因為使用較少記憶體時，JVM 會執行更多處理。建議您調校選項，以平衡記憶體和效能需求。

這些範例組態合併更新使用以下 JVM 選項：

-XX:+UseSerialGC

指定 將序列垃圾回收用於 JVM 堆積空間。序列垃圾收集器速度較慢，但比其他 JVM 垃圾收集實作使用的記憶體較少。

-XX:TieredStopAtLevel=1

指示 JVM 使用一次 Java just-in-time(JIT) 編譯器。由於 JIT 編譯程式碼使用裝置記憶體中的空間，因此使用 JIT 編譯器多次耗用比單一編譯更多的記憶體。

-XmxNNm

設定最大 JVM 堆積大小。

Important

設定過小的堆積大小上限可能會導致效能變慢或out-of-memory錯誤。建議您先測量目前的堆積用量，再使用 **-XmxNNm**選項設定大小上限。使用 JVM 選項設定您

的 `-XX:NativeMemoryTracking=detail` JVM。然後，使用 [jcmd 公用程式](#) 中的 `VM.native_memory` 命令請求來測量您目前的堆積用量。

如果堆積的測量不是選項，請使用 `-Xmx64m` 作為開始值，將堆積大小限制為 64 MB。然後，您可以逐步減少堆積大小上限。對於最小記憶體配置，請使用 `-Xmx32m` 作為起始值，將堆積大小限制為 32 MB。

您可以根據實際需求來增加或減少 `-Xmx` 值；不過，強烈建議不要將堆積大小上限設定為低於 16 MB。根據部署到核心裝置的外掛程式元件，所需的 JVM 堆積大小也可能隨著時間而不同。如果最大堆積大小對您的環境而言太低，則 AWS IoT Greengrass Core 軟體可能會因為記憶體不足而遇到非預期錯誤。如果您因為記憶體不足而遇到效能較慢或發生錯誤，請還原為已知良好的設定。例如，如果您的正常遞交堆積大小為 41428KB，請使用 `-Xmx40m` 來稍微限制堆積用量。

-Xint

指示 JVM 不使用 just-in-time (JIT) 編譯器。反之，JVM 會以僅解譯模式執行。此模式比執行 JIT 編譯程式碼慢（在低端系統上部署可能慢 20 倍）；不過，編譯的程式碼不會使用任何記憶體空間。

如需建立組態合併更新的資訊，請參閱 [更新元件組態](#)。

設定執行元件的使用者

AWS IoT Greengrass 核心軟體可以系統使用者身分執行元件程序，並將與執行軟體程序不同的程序分組。這可提高安全性，因為您可以以根或管理員使用者身分執行 AWS IoT Greengrass 核心軟體，而無需將這些許可授予在核心裝置上執行的元件。

下表指出 AWS IoT Greengrass Core 軟體可以以您指定的使用者身分執行的元件類型。如需詳細資訊，請參閱 [元件類型](#)。

元件類型	設定元件使用者
Huleus	 否

元件類型	設定元件使用者
外掛程式	 否
一般	 是
Lambda（非容器化）	 是
Lambda（容器化）	 是

您必須先建立元件使用者，才能在部署組態中指定它。在 Windows 型裝置上，您還必須將使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的登入資料管理員執行個體中。如需詳細資訊，請參閱[在 Windows 裝置上設定元件使用者](#)。

當您在 Linux 型裝置上設定元件使用者時，您也可以選擇性地指定群組。您可以指定使用者和群組，並以冒號(:) 分隔，格式如下：*user:group*。如果您未指定群組，則 AWS IoT Greengrass Core 軟體會預設為使用者的主要群組。您可以使用名稱或 ID 來識別使用者和群組。

在 Linux 型裝置上，您也可以以不存在的系統使用者身分執行元件，也稱為未知使用者，以提高安全性。Linux 程序可以發出由相同使用者執行的任何其他程序的訊號。未知使用者不會執行其他程序，因此您可以使用未知使用者身分執行元件，以防止元件向核心裝置上的其他元件發出訊號。若要以未知使用者身分執行元件，請指定核心裝置上不存在的使用者 ID。您也可以指定不存在的群組 ID，以做為未知群組執行。

您可以為每個元件和每個核心裝置設定使用者。

- 為元件設定

您可以設定每個元件，以與該元件特定的使用者一起執行。當您建立部署時，您可以為該元件 `runWith` 組態中的每個元件指定使用者。如果您設定元件，AWS IoT Greengrass 則 Core 軟體會以指定的使用者身分執行元件。否則，預設會以您為核心裝置設定的預設使用者身分執行元件。如需在部署組態中指定元件使用者的詳細資訊，請參閱 中的 [runWith](#) 組態參數 [建立部署](#)。

- 設定核心裝置的預設使用者

您可以設定 AWS IoT Greengrass Core 軟體用來執行元件的預設使用者。AWS IoT Greengrass 核心軟體執行元件時，會檢查您是否為該元件指定使用者，並使用它來執行元件。如果元件未指定使用者，則 AWS IoT Greengrass Core 軟體會以您為核心裝置設定的預設使用者身分執行元件。如需詳細資訊，請參閱 [設定預設元件使用者](#)。

 Note

在 Windows 型裝置上，您必須至少指定預設使用者才能執行元件。

在 Linux 型裝置上，如果您未將使用者設定為執行元件，則適用下列考量：

- 如果您執行 AWS IoT Greengrass Core 軟體做為根，則軟體不會執行元件。如果您以根身分執行，則必須指定預設使用者來執行元件。
- 如果您以非根使用者身分執行 AWS IoT Greengrass Core 軟體，則軟體會以該使用者身分執行元件。

主題

- [在 Windows 裝置上設定元件使用者](#)
- [設定預設元件使用者](#)

在 Windows 裝置上設定元件使用者

在 Windows 型裝置上設定元件使用者

1. 在裝置上的 LocalSystem 帳戶中建立元件使用者。

```
net user /add component-user password
```

2. 使用 [Microsoft 的 PsExec 公用程式](#)，將元件使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。

```
psexec -s cmd /c cmdkey /generic:component-user /user:component-user /pass:password
```

Note

在 Windows 型裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將元件使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

設定預設元件使用者

您可以使用部署來設定核心裝置上的預設使用者。在此部署中，您會更新 [nucleus 元件組態](#)。

Note

您也可以在使用 `--component-default-user` 選項安裝 AWS IoT Greengrass Core 軟體時設定預設使用者。如需詳細資訊，請參閱 [安裝 AWS IoT Greengrass Core 軟體](#)。

[建立 部署](#)，指定 `aws.greengrass.Nucleus` 元件的下列組態更新。

Linux

```
{
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

```
}
```

Note

您指定的使用者必須存在，且此使用者的使用者名稱和密碼必須存放在 Windows 裝置上的 LocalSystem 帳戶的登入資料管理員執行個體中。如需詳細資訊，請參閱[在 Windows 裝置上設定元件使用者](#)。

下列範例定義以 Linux 為基礎的裝置的部署，將設定為ggc_user預設使用者，並將 ggc_group設定為預設群組。merge 組態更新需要序列化 JSON 物件。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}\"
      }
    }
  }
}
```

設定元件的系統資源限制

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.4.0 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

您可以設定每個元件程序可在核心裝置上使用的 CPU 和 RAM 用量上限。

下表顯示支援系統資源限制的元件類型。如需詳細資訊，請參閱[元件類型](#)。

元件類型	設定系統資源限制
Huleus	 否
外掛程式	 否
一般	 是
Lambda（非容器化）	 是
Lambda（容器化）	 否

 Important

當您在 [Docker 容器中執行 AWS IoT Greengrass Core 軟體時](#)，不支援系統資源限制。

您可以為每個元件和每個核心裝置設定系統資源限制。

- 為元件設定

您可以使用該元件特定的系統資源限制來設定每個元件。建立部署時，您可以指定部署中每個元件的系統資源限制。如果元件支援系統資源限制，則 AWS IoT Greengrass Core 軟體會將限制套用至元件的程序。如果您未指定元件的系統資源限制，則 AWS IoT Greengrass Core 軟體會使用您為核心裝置設定的任何預設值。如需詳細資訊，請參閱[建立部署](#)。

- 設定核心裝置的預設值

您可以設定 AWS IoT Greengrass Core 軟體套用至支援這些限制之元件的預設系統資源限制。當 AWS IoT Greengrass Core 軟體執行元件時，它會套用您為該元件指定的系統資源限制。如果該元件未指定系統資源限制，則 AWS IoT Greengrass Core 軟體會套用您為核心裝置設定的預設系統資源限制。如果您未指定預設系統資源限制，AWS IoT Greengrass 則 Core 軟體預設不會套用任何系統資源限制。如需詳細資訊，請參閱[設定預設系統資源限制](#)。

設定預設系統資源限制

您可以部署 [Greengrass 核心元件](#)，以設定核心裝置的預設系統資源限制。若要設定預設系統資源限制，請[建立部署](#)來指定aws.greengrass.Nucleus元件的下列組態更新。

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

下列範例會定義將 CPU 時間限制設定為 的部署2，相當於具有 4 個 CPU 核心之裝置上的 50% 用量。此範例也會將記憶體用量設定為 100 MB。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":{\"cpus\":2,\"memory\":102400}}}"
      }
    }
  }
}
```

```
}
```

連線至連接埠 443 或透過網路代理

AWS IoT Greengrass 核心裝置 AWS IoT Core 使用 MQTT 訊息通訊協定搭配 TLS 用戶端身分驗證與通訊。根據慣例，透過 TLS 的 MQTT 使用連接埠 8883。不過，做為安全措施之用，受限環境可能會將傳入和傳出流量限制於小範圍的 TCP 連接埠。例如，企業防火牆可能會針對 HTTPS 流量開放連接埠 443，但關閉較不常用之通訊協定的其他連接埠，例如 MQTT 流量的連接埠 8883。其他限制性環境可能需要所有流量通過代理才能連線至網際網路。

Note

執行 Greengrass [核心元件 v2.0.3 和更早版本的 Greengrass](#) 核心裝置使用連接埠 8443 連接到 AWS IoT Greengrass 資料平面端點。這些裝置必須能夠在連接埠 8443 上連線至此端點。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

若要在這些案例中啟用通訊，AWS IoT Greengrass 提供下列組態選項：

- 透過連接埠 443 的 MQTT 通訊。如果您的網路允許連線至連接埠 443，您可以將 Greengrass 核心裝置設定為將連接埠 443 用於 MQTT 流量，而不是預設連接埠 8883。這可以直接連線至連接埠 443，或透過網路代理伺服器連線。與使用憑證型用戶端身分驗證的預設組態不同，連接埠 443 上的 MQTT 會使用[裝置服務角色](#)進行身分驗證。

如需詳細資訊，請參閱[透過連接埠 443 設定 MQTT](#)。

- 透過連接埠 443 的 HTTPS 通訊。Core AWS IoT Greengrass 軟體預設會透過連接埠 8443 傳送 HTTPS 流量，但您可以將其設定為使用連接埠 443。AWS IoT Greengrass 會使用[應用程式層通訊協定網路](#) (ALPN) TLS 延伸來啟用此連線。如同預設組態，連接埠 443 上的 HTTPS 使用憑證型用戶端身分驗證。

Important

若要使用 ALPN 並透過連接埠 443 啟用 HTTPS 通訊，您的核心裝置必須執行 Java 8 更新 252 或更新版本。Java 第 9 版及更新版本的所有更新也支援 ALPN。

如需詳細資訊，請參閱[透過連接埠 443 設定 HTTPS](#)。

- 透過網路代理的連線。您可以設定網路代理伺服器做為連線至 Greengrass 核心裝置的媒介。AWS IoT Greengrass 支援 HTTP 和 HTTPS 代理的基本身分驗證。

Greengrass 核心裝置必須執行 [Greengrass 核 v2.5.0](#) 或更新版本，才能使用 HTTPS 代理。

Core AWS IoT Greengrass 軟體會透過 ALL_PROXY、HTTPS_PROXY、和 HTTP_PROXY 環境變數，將代理組態傳遞至元件。元件必須使用這些設定來透過代理連線。元件使用常見程式庫（例如 boto3、cURL 和 python requests 套件），這些程式庫預設會使用這些環境變數進行連線。如果元件也指定這些環境變數，則 AWS IoT Greengrass 不會覆寫它們。

如需詳細資訊，請參閱[設定網路代理](#)。

透過連接埠 443 設定 MQTT

您可以在現有的核心裝置上，或在新的核心裝置上安裝 AWS IoT Greengrass 核心軟體時，透過連接埠 443 設定 MQTT。

主題

- [在現有核心裝置上透過連接埠 443 設定 MQTT](#)
- [在安裝期間透過連接埠 443 設定 MQTT](#)

在現有核心裝置上透過連接埠 443 設定 MQTT

您可以使用部署，在單一核心裝置或一組核心裝置上透過連接埠 443 設定 MQTT。在此部署中，您會更新 [nucleus 元件](#) 組態。當您更新其 mqtt 組態時，核會重新啟動。

若要透過連接埠 443 設定 MQTT，[請建立 部署](#)來指定 `aws.greengrass.Nucleus` 元件的下列組態更新。

```
{
  "mqtt": {
    "port": 443
  }
}
```

下列範例定義透過連接埠 443 設定 MQTT 的部署。merge 組態更新需要序列化 JSON 物件。

```
{
```

```
"components": {
  "aws.greengrass.Nucleus": {
    "version": "2.16.1",
    "configurationUpdate": {
      "merge": "{\"mqtt\":{\"port\":443}}}"
    }
  }
}
```

在安裝期間透過連接埠 443 設定 MQTT

您可以在核心裝置上安裝 AWS IoT Greengrass Core 軟體時，透過連接埠 443 設定 MQTT。使用 `--init-config` 安裝程式引數透過連接埠 443 設定 MQTT。您可以在使用[手動佈建](#)、[機群佈建](#)或[自訂佈建](#)安裝時指定此引數。

透過連接埠 443 設定 HTTPS

此功能需要 v2[Greengrass 核](#).0.4 或更新版本。

您可以在現有核心裝置上，或在新核心裝置上安裝 AWS IoT Greengrass 核心軟體時，透過連接埠 443 設定 HTTPS。

主題

- [在現有核心裝置上透過連接埠 443 設定 HTTPS](#)
- [在安裝期間透過連接埠 443 設定 HTTPS](#)

在現有核心裝置上透過連接埠 443 設定 HTTPS

您可以使用部署，在單一核心裝置或一組核心裝置上透過連接埠 443 設定 HTTPS。在此部署中，您會更新 [nucleus 元件](#) 組態。

若要透過連接埠 443 設定 HTTPS，[請建立指定下列元件組態更新的部署](#)。

`aws.greengrass.Nucleus`

```
{
  "greengrassDataPlanePort": 443
}
```

下列範例定義透過連接埠 443 設定 HTTPS 的部署。merge 組態更新需要序列化 JSON 物件。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.16.1",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

在安裝期間透過連接埠 443 設定 HTTPS

您可以在核心裝置上安裝 AWS IoT Greengrass Core 軟體時，透過連接埠 443 設定 HTTPS。使用 `--init-config` 安裝程式引數透過連接埠 443 設定 HTTPS。您可以在使用 [手動佈建](#)、[機群佈建](#) 或 [自訂佈建](#) 安裝時指定此引數。

設定網路代理

遵循本節中的程序，設定 Greengrass 核心裝置透過 HTTP 或 HTTPS 網路代理連線到網際網路。如需核心裝置使用的端點和連接埠的詳細資訊，請參閱 [允許透過代理或防火牆的裝置流量](#)。

Important

如果您的核心裝置執行比 v2.4.0 更舊的 [Greengrass 核](#) 版本，您裝置的角色必須允許下列許可才能使用網路代理：

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

這是必要的，因為裝置會使用字符交換服務的 AWS 登入資料來驗證 MQTT 連線 AWS IoT。裝置使用 MQTT 從接收和安裝部署 AWS 雲端，因此除非您為其角色定義這些許可，否則您的裝置將無法運作。裝置通常會使用 X.509 憑證來驗證 MQTT 連線，但裝置無法使用代理來驗證。

如需如何設定裝置角色的詳細資訊，請參閱 [授權核心裝置與 AWS 服務互動](#)。

主題

- [在現有核心裝置上設定網路代理](#)
- [在安裝期間設定網路代理](#)
- [讓核心裝置信任 HTTPS 代理](#)
- [networkProxy 物件](#)

在現有核心裝置上設定網路代理

您可以使用部署在單一核心裝置或一組核心裝置上設定網路代理。在此部署中，您會更新 [nucleus 元件](#) 組態。當您更新其 networkProxy 組態時，核會重新啟動。

若要設定網路代理，請為合併下列組態更新的 aws.greengrass.Nucleus 元件 [建立部署](#)。此組態更新包含 [networkProxy 物件](#)。

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

下列範例會定義可設定網路代理的部署。merge 組態更新需要序列化 JSON 物件。

```
{
  "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\"}}}"
      }
    }
  }
}
```

在安裝期間設定網路代理

您可以在核心裝置上安裝 AWS IoT Greengrass Core 軟體時設定網路代理。使用 `--init-config` 安裝程式引數來設定網路代理。您可以在使用[手動佈建](#)、[機群佈建](#)或[自訂佈建](#)安裝時指定此引數。

讓核心裝置信任 HTTPS 代理

當您將核心裝置設定為使用 HTTPS 代理時，必須將代理伺服器憑證鏈新增至核心裝置的，讓它信任 HTTPS 代理。否則，核心裝置可能會在嘗試透過代理路由流量時遇到錯誤。將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證檔案。

讓核心裝置信任 HTTPS 代理

1. 尋找核心裝置上的 Amazon 根 CA 憑證檔案。

- 如果您使用[自動佈建](#)安裝 AWS IoT Greengrass Core 軟體，Amazon 根 CA 憑證檔案會存在於 `/greengrass/v2/rootCA.pem`。
- 如果您使用[手動](#)或[機群佈建](#)安裝 AWS IoT Greengrass Core 軟體，Amazon 根 CA 憑證檔案可能存在於 `/greengrass/v2/AmazonRootCA1.pem`。

如果這些位置不存在 Amazon 根 CA 憑證，請檢查 `system.rootCaPath` 屬性/`greengrass/v2/config/effectiveConfig.yaml`以尋找其位置。

2. 將代理伺服器 CA 憑證檔案的內容新增至 Amazon 根 CA 憑證檔案。

下列範例顯示新增至 Amazon 根 CA 憑證檔案的代理伺服器 CA 憑證。

```
-----BEGIN CERTIFICATE-----
MIIEFTCCA v2gAwIQWgIVAMHSAzWG/5YVRYtRQ0xXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBjbmuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/n07f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/1ND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimGF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QTPHRh8jrdkGA1UEChMGDV3QQDExBKw
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJioblXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
```

```
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

networkProxy 物件

使用 networkProxy 物件，指定網路代理的相關資訊。此物件包含下列資訊：

noProxyAddresses

(選用) 從代理中豁免的 IP 地址或主機名稱的逗號分隔清單。

proxy

要連線的代理。此物件包含下列資訊：

url

代理伺服器的 URL，格式為 `scheme://userinfo@host:port`。

- `scheme` – 結構描述，必須是 `http` 或 `https`。

Important

Greengrass 核心裝置必須執行 [Greengrass 核 v2.5.0](#) 或更新版本，才能使用 HTTPS 代理。

如果您設定 HTTPS 代理，則必須將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證。如需詳細資訊，請參閱[讓核心裝置信任 HTTPS 代理](#)。

- `userinfo` – (選用) 使用者名稱和密碼資訊。如果您在 `url` 中指定此資訊，Greengrass 核心裝置會忽略 `username` 和 `password` 欄位。
- `host` – 代理伺服器的主機名稱或 IP 地址。
- `port` – (選用) 連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：
 - `http` – 80
 - `https` – 443

username

(選用) 驗證代理伺服器的使用者名稱。

password

(選用) 驗證代理伺服器的密碼。

使用私有 CA 簽署的裝置憑證

如果您使用自訂私有憑證授權機構 (CA)，您必須將 Greengrass 核心 `greengrassDataPlaneEndpoint` 設定為 `iotdata`。您可以使用 `--init-config` [安裝程式引數](#) 在部署或安裝期間設定此選項。

您可以自訂裝置連線的 Greengrass 資料平面端點。您可以將此組態選項設定為 `iotdata`，將 Greengrass 資料平面端點設定為與 IoT 資料端點相同的端點，您可以使用 `iotDataEndpoint` 來指定 `iotDataEndpoint`。

設定 MQTT 逾時和快取設定

在 AWS IoT Greengrass 環境中，元件可以使用 MQTT 與通訊 AWS IoT Core。Core AWS IoT Greengrass 軟體會管理元件的 MQTT 訊息。當核心裝置失去與 AWS 雲端的連線時，軟體會快取 MQTT 訊息，以便在連線還原時稍後重試。您可以設定訊息逾時和快取大小等設定。如需詳細資訊，請參閱 [Greengrass 核心元件](#) 的 `mqtt` 和 `mqtt.spooler` 組態參數。

AWS IoT Core 對其 MQTT 訊息代理程式強加服務配額。這些配額可能適用於您在核心裝置和 AWS IoT Core 之間傳送的訊息。如需詳細資訊，請參閱 [AWS IoT Core 訊息中介裝置服務配額](#) 中的 [AWS IoT Core 訊息中介裝置服務配額](#) 一般參考。

在 IPv6 網路上設定 Greengrass Nucleus

Greengrass Nucleus 透過 [Greengrass APIs](#) 與 AWS IoT Core 通訊。Greengrass APIs 雙堆疊環境中支援 IPv6。

若要啟用 IPv6 的雙堆疊端點：

- 將系統屬性 `aws.useDualstackEndpoint=true`、和 `java.net.preferIPv6Addresses=true` 新增至 `jvmOptions`
- 將 `s3EndpointType` 設定為 `DUALSTACK`

在 [部署](#) 期間設定此選項，或使用 `--init-config` [安裝程式引數](#) 手動佈建。如需詳細資訊，請參閱 [使用 Amazon S3 雙堆疊端點](#)。

Example 部署的程式碼：

```
{
  "jvmOptions": "-Daws.useDualstackEndpoint=true",
  "s3EndpointType": "DUALSTACK"
```

```
}
```

Example **config.yaml** 透過手動佈建：

```
---
system:
  ...
services:
  aws.greengrass.Nucleus:
    ...
    configuration:
      ...
      jvmOptions: "-Daws.useDualstackEndpoint=true -Djava.net.preferIPv6Addresses=true"
      s3EndpointType: "DUALSTACK"
```

更新 AWS IoT Greengrass 核心軟體 (OTA)

AWS IoT Greengrass 核心軟體包含 [Greengrass 核元件](#) 和其他選用元件，您可以將這些元件部署到您的裝置，以執行軟體的 over-the-air (OTA) 更新。此功能內建於 AWS IoT Greengrass 核心軟體。

OTA 更新可讓您更有效率地執行下列作業：

- 修復安全漏洞。
- 解決軟體穩定性問題。
- 部署新功能和改良的功能。

主題

- [要求](#)
- [核心裝置的考量事項](#)
- [Greengrass 核更新行為](#)
- [執行 OTA 更新](#)

要求

下列需求適用於部署 AWS IoT Greengrass 核心軟體的 OTA 更新：

- Greengrass 核心裝置必須連線至 AWS 雲端 才能接收部署。

- Greengrass 核心裝置必須正確設定和佈建憑證和金鑰，以便使用 AWS IoT Core 和 進行身分驗證 AWS IoT Greengrass。
- AWS IoT Greengrass 核心軟體必須設定為系統服務並執行。如果您從 JAR 檔案 執行 核，OTA 更新將無法運作Greengrass.jar。如需詳細資訊，請參閱[將 Greengrass 核設定為系統服務](#)。

核心裝置的考量事項

執行 OTA 更新之前，請注意您更新的核心裝置及其連線的用戶端裝置的影響：

- Greengrass 核會關閉。
- 在核心裝置上執行的所有元件也會關閉。如果這些元件寫入本機資源，除非正確關閉，否則這些資源可能會處於不正確的狀態。元件可以使用[程序間通訊](#)來告知 nucleus 元件延遲更新，直到清除其使用的資源為止。
- 當核心元件關閉時，核心裝置會失去與 AWS 雲端 和本機裝置的連線。關閉時，核心裝置不會從用戶端裝置路由訊息。
- 做為元件執行的長期 Lambda 函數會遺失其動態狀態資訊，並捨棄所有待處理的工作。

Greengrass 核更新行為

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將新修補程式版本的 AWS 公有元件部署到核心裝置。有些自動更新，例如 核更新，可能會導致您的裝置意外重新啟動。

當 [Greengrass 核子元件](#) 的版本變更時，包含您裝置上的核子和所有其他元件 AWS IoT Greengrass 的核心軟體會開始套用變更。由於 核元件更新時[對核心裝置的影響](#)，因此您可能想要控制何時將新的核修補程式版本部署至您的裝置。若要這樣做，您必須在部署中直接包含 Greengrass 核元件。直接包含元件表示您在部署組態中包含該元件的特定版本，且不依賴元件相依性將該元件部署至您的裝置。如需在元件配方中定義相依性的詳細資訊，請參閱[配方格式](#)。

檢閱下表，根據您的動作和部署組態了解 Greengrass 核元件的更新行為。

動作	部署組態	Nucleus 更新行為
將新裝置新增至現有部署以目標為目標的物件群組，而不修改部署。	部署不會直接包含 Greengrass 核。	在新裝置上，安裝符合所有元件相依性需求的最新版 核。

動作	部署組態	Nucleus 更新行為
	部署直接包含至少一個 AWS 提供的元件，或包含取決於 AWS 提供的元件或 Greengrass 核的自訂元件。	在現有裝置上，不會更新已安裝的核版本。
將新裝置新增至現有部署以目標為目標的物件群組，而不修改部署。	部署直接包含特定版本的 Greengrass 核。	在新裝置上，安裝指定的 nucleus 版本。 在現有裝置上，不會更新已安裝的核版本。
建立新的部署或修改現有的部署。	部署不會直接包含 Greengrass 核。 部署直接包含至少一個 AWS 提供的元件，或包含取決於 AWS 提供的元件或 Greengrass 核的自訂元件。	在所有目標裝置上，會安裝符合所有元件相依性需求的最新版核，包括您新增至目標物件群組的任何新裝置。
建立新的部署或修改現有的部署。	部署直接包含特定版本的 Greengrass 核。	在所有目標裝置上，安裝指定的核版本，包括您新增至目標物件群組的任何新裝置。

執行 OTA 更新

若要執行 OTA 更新，[請建立包含核元件和要安裝之版本的部署](#)。 ???

解除安裝 AWS IoT Greengrass Core 軟體

您可以解除安裝 AWS IoT Greengrass Core 軟體，將其從您不想用作 Greengrass 核心裝置的裝置中移除。您也可以使用這些步驟來清除失敗的安裝。

解除安裝 AWS IoT Greengrass Core 軟體

1. 如果您以系統服務方式執行軟體，則必須停止、停用和移除服務。根據您的作業系統執行下列命令。

Linux

1. 停止 服務。

```
sudo systemctl stop greengrass.service
```

2. 停用服務。

```
sudo systemctl disable greengrass.service
```

3. 移除 服務。

```
sudo rm /etc/systemd/system/greengrass.service
```

4. 確認已刪除服務。

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

Note

您必須以管理員身分執行命令提示，才能執行這些命令。

1. 停止 服務。

```
sc stop "greengrass"
```

2. 停用服務。

```
sc config "greengrass" start=disabled
```

3. 移除 服務。

```
sc delete "greengrass"
```

4. 重新啟動裝置。

Windows (PowerShell)

Note

您必須以管理員身分執行 PowerShell，才能執行這些命令。

1. 停止 服務。

```
Stop-Service -Name "greengrass"
```

2. 停用服務。

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. 移除 服務。

- 對於 PowerShell 6.0 和更新版本：

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

- 對於 6.0 之前的 PowerShell 版本：

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item  
-Force -Verbose
```

4. 重新啟動裝置。

2. 從裝置移除根資料夾。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為根資料夾的路徑。

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. 從 AWS IoT Greengrass 服務中刪除核心裝置。此步驟會從 移除核心裝置的狀態資訊 AWS 雲端。如果您計劃將 AWS IoT Greengrass Core 軟體重新安裝至具有相同名稱的核心裝置，請務必完成此步驟。
 - 若要從 AWS IoT Greengrass 主控台刪除核心裝置，請執行下列動作：
 - a. 導覽至 [AWS IoT Greengrass 主控台](#)。
 - b. 選擇核心裝置。
 - c. 選擇要刪除的核心裝置。
 - d. 選擇 刪除。
 - e. 在確認模式中，選擇刪除。
 - 若要使用 刪除核心裝置 AWS Command Line Interface，請使用 [DeleteCoreDevice](#) 操作。執行下列命令，並以核心裝置的名稱取代 *MyGreengrassCore*。

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 教學課程

AWS IoT Greengrass 是一項服務，可讓您在邊緣裝置上執行 AWS Lambda 函數、機器學習模型和其他程式碼。這可讓您在本地處理資料，減少延遲和頻寬成本，同時維持與雲端的安全通訊。

您可以完成下列教學課程，以了解 AWS IoT Greengrass V2 及其功能。

主題

- [教學課程影片](#)
- [教學課程：開發可延遲元件更新的 Greengrass 元件](#)
- [教學課程：透過 MQTT 與本地 IoT 裝置互動](#)
- [教學課程：使用信任平台模組 \(TPM\) 保護 Greengrass Nucleus](#)
- [教學課程：使用信任平台模組 \(TPM\) 保護 AWS IoT Greengrass Nucleus Lite](#)
- [教學課程：開始使用 SageMaker AI Edge Manager](#)
- [教學課程：使用 TensorFlow Lite 執行範例影像分類推論](#)
- [教學課程：使用 TensorFlow Lite 從相機執行影像分類推論範例](#)

教學課程影片

影片

下列清單包含的影片教學課程 AWS IoT Greengrass V2。

- [AWS IoT Greengrass nucleus lite 教學課程：入門](#)

2025 年 4 月 10 日 – 在本教學課程中，您將了解 Greengrass nucleus lite 以及何時為 AWS IoT 您的裝置選取它。您將了解如何安裝 nucleus lite 並將元件部署到其中。

- [AWS IoT Greengrass V2 教學課程：入門](#)

2021 年 12 月 16 日 – 開始使用 AWS IoT Greengrass V2。在本教學課程中，您將了解如何安裝 Greengrass 並完成您的第一個部署。

- [AWS IoT Greengrass V2 使用本地偵錯主控台進行元件開發](#)

2023 年 8 月 21 日 – 了解如何使用 Greengrass V2 本地偵錯主控台快速開發和偵錯元件。

- [將 MQTT 用戶端裝置連接至 AWS IoT Greengrass V2](#)

2023 年 8 月 2 日 – 在本教學課程中，您將了解如何設定核心裝置與透過 MQTT 連線至核心裝置的本機 AWS IoT 裝置互動。

- [AWS IoT Greengrass V2 教學課程：使用 MQTT 應用程式](#)

2021 年 12 月 16 日 – 建立、測試和部署 AWS IoT Greengrass V2 原生元件，以透過發佈和訂閱 MQTT AWS IoT Core。

教學課程：開發可延遲元件更新的 Greengrass 元件

您可以完成本教學課程，以開發延遲over-the-air部署更新的元件。當您部署更新至裝置時，您可能會想要根據條件延遲更新，例如下列項目：

- 裝置電池電量低。
- 裝置正在執行無法中斷的程序或任務。
- 裝置具有有限或昂貴的網際網路連線。

Note

元件是在 AWS IoT Greengrass 核心裝置上執行的軟體模組。元件可讓您將複雜的應用程式建立和管理為分散的建置區塊，以便從一個 Greengrass 核心裝置重複使用。

在此教學課程中，您將執行下列操作：

1. 在開發電腦上安裝 Greengrass 開發套件 CLI (GDK CLI)。GDK CLI 提供可協助您開發自訂 Greengrass 元件的功能。
2. 開發 Hello World 元件，在核心裝置的電池電量低於閾值時延遲元件更新。此元件訂閱使用 [SubscribeToComponentUpdates](#) IPC 操作來更新通知。收到通知時，它會檢查電池電量是否低於可自訂的閾值。如果電池電量低於閾值，則會使用 [DeferComponentUpdate](#) IPC 操作將更新延遲 30 秒。您可以使用 GDK CLI 在開發電腦上開發此元件。

Note

此元件會從您在核心裝置上建立的檔案讀取電池電量，以模擬真正的電池，因此您可以在核心裝置上完成此教學課程，而無需電池。

3. 將該元件發佈至 AWS IoT Greengrass 服務。
4. 將該元件從 部署 AWS 雲端 到 Greengrass 核心裝置以進行測試。然後，您可以修改核心裝置上的虛擬電池電量，並建立其他部署，以查看核心裝置在電池電量低時如何延遲更新。

您可以預期在本教學課程中花費 20-30 分鐘。

先決條件

為了完成本教學，您需要以下項目：

- AWS 帳戶：如果您沒有帳戶，請參閱 [設定 AWS 帳戶](#)。
- 具有管理員許可的 AWS Identity and Access Management (IAM) 使用者。
- 具有網際網路連線的 Greengrass 核心裝置。如需如何設定核心裝置的詳細資訊，請參閱[設定 AWS IoT Greengrass 核心裝置](#)。
- 核心裝置上所有使用者已安裝 [Python](#) 3.6 或更新版本，並已新增至PATH環境變數。在 Windows 上，您也必須為所有使用者安裝適用於 Windows 的 Python Launcher。

Important

在 Windows 中，Python 預設不會為所有使用者安裝。安裝 Python 時，您必須自訂安裝以設定安裝，AWS IoT Greengrass 核心軟體才能執行 Python 指令碼。例如，如果您使用圖形 Python 安裝程式，請執行下列動作：

1. 選取安裝所有使用者的啟動器（建議）。
2. 選擇 Customize installation。
3. 選擇 Next。
4. 選取 Install for all users。
5. 選取 Add Python to environment variables。
6. 選擇 Install (安裝)。

如需詳細資訊，請參閱 [Python 3 文件中的在 Windows 上使用 Python](#)。

- 具有網際網路連線的 Windows、macOS 或類似 Unix 的開發電腦。
- 您的開發電腦上已安裝 [Python](#) 3.6 或更新版本。
- 安裝在開發電腦上的 [Git](#)。

- AWS Command Line Interface (AWS CLI) 在開發電腦上安裝並設定 憑證。如需詳細資訊，請參閱AWS Command Line Interface 《使用者指南》中的[安裝、更新和解除安裝 AWS CLI](#) 和[設定 AWS CLI](#)。

 Note

如果您使用 Raspberry Pi 或其他 32 位元 ARM 裝置，則 install AWS CLI V1。AWS CLI V2 不適用於 32 位元 ARM 裝置。如需詳細資訊，請參閱[安裝、更新和解除安裝 AWS CLI 版本 1](#)。

步驟 1：安裝 Greengrass 開發套件 CLI

[Greengrass 開發套件 CLI \(GDK CLI\)](#) 提供可協助您開發自訂 Greengrass 元件的功能。您可以使用 GDK CLI 來建立、建置和發佈自訂元件。

如果您尚未在開發電腦上安裝 GDK CLI，請完成下列步驟來安裝它。

安裝最新版本的 GDK CLI

1. 在開發電腦上，執行下列命令，從其 [GitHub 儲存庫](#) 安裝最新版本的 GDK CLI。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

2. 執行下列命令來驗證 GDK CLI 是否已成功安裝。

```
gdk --help
```

如果找不到 gdk 命令，請將其資料夾新增至 PATH。

- 在 Linux 裝置上，將 `/home/MyUser/.local/bin` 新增至 PATH，並以您的使用者名稱取代 `MyUser`。
- 在 Windows 裝置上，將 `PythonPath\Scripts` 新增至 PATH，並將 `PythonPath` 取代為您裝置上的 Python 資料夾路徑。

步驟 2：開發延遲更新的元件

在本節中，您會在 Python 中開發 Hello World 元件，該元件會在核心裝置的電池電量低於部署元件時設定的閾值時延遲元件更新。在此元件中，您會在 AWS IoT Device SDK v2 for Python 中使用[程序間通訊 \(IPC\) 界面](#)。您可以使用 [SubscribeToComponentUpdates](#) IPC 操作，在核心裝置收到部署時接收通知。然後，您可以使用 [DeferComponentUpdate](#) IPC 操作，根據裝置的電池電量延遲或確認更新。

開發延遲更新的 Hello World 元件

1. 在開發電腦上，為元件原始程式碼建立資料夾。

```
mkdir com.example.BatteryAwareHelloWorld
cd com.example.BatteryAwareHelloWorld
```

2. 使用文字編輯器來建立名為 `gdk-config.json` 的檔案。GDK CLI 會從名為 [GDK CLI 組態檔案](#) 讀取 `gdk-config.json`，以建置和發佈元件。此組態檔案存在於元件資料夾的根目錄中。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano gdk-config.json
```

將下列 JSON 複製到 檔案。

- 以您的名稱取代 *Amazon*。
- 將 *us-west-2* 取代為核心裝置運作 AWS 區域 所在的 。GDK CLI 會在此發佈元件 AWS 區域。
- 將 *greengrass-component-artifacts* 取代為要使用的 S3 儲存貯體字首。當您使用 GDK CLI 發佈元件時，GDK CLI 會使用下列格式，將元件的成品上傳至 S3 儲存貯體，其名稱是從此值 AWS 區域、和您的 AWS 帳戶 ID 形成：*bucketPrefix-region-accountId*。

例如，如果您指定 **greengrass-component-artifacts** 和 **us-west-2**，且 AWS 帳戶 ID 為 **123456789012**，則 GDK CLI 會使用名為 `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"
}

```

組態檔案會指定下列項目：

- 當 GDK CLI 將 Greengrass 元件發佈至雲端服務時要使用的 AWS IoT Greengrass 版本。
NEXT_PATCH指定在 AWS IoT Greengrass 雲端服務中可用的最新版本之後選擇下一個修補程式版本。如果元件在 AWS IoT Greengrass 雲端服務中還沒有版本，GDK CLI 會使用 1.0.0。
 - 元件的建置系統。當您使用zip建置系統時，GDK CLI 會將元件的來源封裝成 ZIP 檔案，該檔案會成為元件的單一成品。
 - AWS 區域 GDK CLI 發佈 Greengrass 元件的。
 - GDK CLI 上傳元件成品的 S3 儲存貯體字首。
3. 使用文字編輯器在名為 的檔案中建立元件原始程式碼main.py。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano main.py
```

將下列 Python 程式碼複製到 檔案。

```

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()

```

此 Python 應用程式會執行下列動作：

- 從您稍後將在核心裝置上建立的虛擬電池電量檔案中讀取核心裝置的電池電量。此虛擬電池電量檔案會模擬真正的電池，因此您可以在沒有電池的核心裝置上完成此教學課程。
- 讀取電池閾值的命令列引數，以及虛擬電池電量檔案的路徑。元件配方會根據組態參數設定這些命令列引數，因此您可以在部署元件時自訂這些值。
- 使用適用於 Python 的 vV2 中的 IPC 用戶端 V2 與 AWS IoT Greengrass Core 軟體通訊。[AWS IoT Device SDK](#)相較於原始 IPC 用戶端，IPC 用戶端 V2 可減少您在自訂元件中使用 IPC 所需的程式碼量。
- 訂閱以使用 [SubscribeToComponentUpdates](#) IPC 操作更新通知。AWS IoT Greengrass 核心軟體會在每次部署前後傳送通知。元件每次收到通知時都會呼叫下列函數。如果通知適用於即將進行的部署，元件會檢查電池電量是否低於閾值。如果電池電量低於閾值，元件會使用 [DeferComponentUpdate](#) IPC 操作將更新延遲 30 秒。否則，如果電池電量未低於閾值，元件會確認更新，因此可以繼續更新。

```

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

Core AWS IoT Greengrass 軟體不會傳送本機部署的更新通知，因此您可以使用 AWS IoT Greengrass 雲端服務來部署此元件來進行測試。

4. 使用文字編輯器在名為 `recipe.json` 或的檔案中建立元件配方 `recipe.yaml`。元件配方定義元件的中繼資料、預設組態參數和平台特定的生命週期指令碼。

JSON

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano recipe.json
```

將下列 JSON 複製到 檔案。

```
{
  "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 awsiot-sdk --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 awsiot-sdk --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

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。


```
nano recipe.yaml
```

將下列 YAML 複製到 檔案。

```
---
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 awsiot-sdk --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 awsiot-sdk --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
```

此配方會指定下列項目：

- 電池閾值的預設組態參數、Linux 核心裝置上的虛擬電池檔案路徑，以及 Windows 核心裝置上的虛擬電池檔案路徑。
- 安裝 AWS IoT Device SDK 最新版本 v2 for Python 的 `install` 生命週期。
- 在 `run` 生命週期中執行 Python 應用程式的 `main.py`。
- 預留位置，例如 `COMPONENT_NAME` 和 `COMPONENT_VERSION`，其中 GDK CLI 會在建置元件配方時取代資訊。

如需元件配方的詳細資訊，請參閱 [AWS IoT Greengrass 元件配方參考](#)。

步驟 3：將元件發佈至 AWS IoT Greengrass 服務

在本節中，您將 Hello World 元件發佈至 AWS IoT Greengrass 雲端服務。在 AWS IoT Greengrass 雲端服務中提供元件之後，您可以將其部署到核心裝置。您可以使用 GDK CLI 將元件從開發電腦發佈至 AWS IoT Greengrass 雲端服務。GDK CLI 會為您上傳元件的配方和成品。

將 Hello World 元件發佈至 AWS IoT Greengrass 服務

1. 執行下列命令，使用 GDK CLI 建置元件。[元件建置命令](#)會根據 GDK CLI 組態檔案建立配方和成品。在此程序中，GDK CLI 會建立包含元件原始碼的 ZIP 檔案。

```
gdk component build
```

您應該會看到類似下列範例的訊息。

```
[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. 執行下列命令，將元件發佈至 AWS IoT Greengrass 雲端服務。[元件發佈命令](#)會將元件的 ZIP 檔案成品上傳至 S3 儲存貯體。然後，它會更新元件配方中 ZIP 檔案的 S3 URI，並將配方上傳至服務 AWS IoT Greengrass。在此程序中，GDK CLI 會檢查 AWS IoT Greengrass 雲端服務中已有哪個版本的 Hello World 元件可用，因此可以選擇該版本之後的下一個修補程式版本。如果元件尚不存在，GDK CLI 會使用版本 1.0.0。

```
gdk component publish
```

您應該會看到類似下列範例的訊息。輸出會告訴您 GDK CLI 建立的元件版本。

```
[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. 從輸出複製 S3 儲存貯體名稱。您稍後會使用儲存貯體名稱，以允許核心裝置從此儲存貯體下載元件成品。
4. (選用) 在 AWS IoT Greengrass 主控台中檢視元件，以確認其已成功上傳。請執行下列操作：

- a. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
- b. 在元件頁面上，選擇我的元件索引標籤，然後選擇 `com.example.BatteryAwareHelloWorld`。

在此頁面上，您可以查看元件的配方和有關元件的其他資訊。

5. 允許核心裝置存取 S3 儲存貯體中的元件成品。

每個核心裝置都有一個[核心裝置 IAM 角色](#)，允許它與 互動 AWS IoT 並將日誌傳送到 AWS 雲端。根據預設，此裝置角色不允許存取 S3 儲存貯體，因此您必須建立並連接允許核心裝置從 S3 儲存貯體擷取元件成品的政策。

如果您裝置的角色已允許存取 S3 儲存貯體，您可以略過此步驟。否則，請建立允許存取的 IAM 政策，並將其連接到角色，如下所示：

- a. 在 [IAM 主控台](#)導覽功能表中，選擇政策，然後選擇建立政策。
- b. 在 JSON 標籤上，用以下政策取代預留位置內容。將 *greengrass-component-artifacts-us-west-2-123456789012* 取代為 GDK CLI 上傳元件成品的 S3 儲存貯體名稱。

例如，如果您 **us-west-2** 在 GDK CLI 組態檔案中指定 **greengrass-component-artifacts** 和 **us-west-2**，且 AWS 帳戶 ID 為 **123456789012**，則 GDK CLI 會使用名為 **greengrass-component-artifacts-us-west-2-123456789012** 的 S3 儲存貯體。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-west-2-123456789012/*"
    }
  ]
}
```

- c. 選擇下一步。

- d. 在政策詳細資訊區段中，針對名稱輸入 **MyGreengrassV2ComponentArtifactPolicy**。
- e. 選擇建立政策。
- f. 在 [IAM 主控台](#) 導覽功能表中，選擇角色，然後選擇核心裝置的角色名稱。您在安裝 AWS IoT Greengrass 核心軟體時指定了此角色名稱。如果您未指定名稱，則預設值為 **GreengrassV2TokenExchangeRole**。
- g. 在許可下，選擇新增許可，然後選擇連接政策。
- h. 在新增許可頁面上，選取您建立 **MyGreengrassV2ComponentArtifactPolicy** 的政策旁的核取方塊，然後選擇新增許可。

步驟 4：在核心裝置上部署和測試元件

在本節中，您將元件部署至核心裝置以測試其功能。在核心裝置上，您可以建立虛擬電池電量檔案來模擬真正的電池。然後，您可以建立其他部署並觀察核心裝置上的元件日誌檔案，以查看元件延遲和確認更新。

部署和測試延遲更新的 Hello World 元件

1. 使用文字編輯器來建立虛擬電池電量檔案。此檔案會模擬真正的電池。
 - 在 Linux 核心裝置上，建立名為 `/home/ggc_user/virtual_battery.json`。執行具有 `sudo` 許可的文字編輯器。
 - 在 Windows 核心裝置上，建立名為 `C:\Users\ggc_user\virtual_battery.json`。以管理員身分執行文字編輯器。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
sudo nano /home/ggc_user/virtual_battery.json
```

將下列 JSON 複製到 檔案。

```
{
  "battery_level": 50
}
```

2. 將 Hello World 元件部署至核心裝置。請執行下列操作：
 - a. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇元件。

- b. 在元件頁面上，選擇我的元件索引標籤，然後選擇 com.example.BatteryAwareHelloWorld。
- c. 在 com.example.BatteryAwareHelloWorld 頁面中，選擇部署。
- d. 從新增至部署中，選擇要修改的現有部署，或選擇建立新的部署，然後選擇下一步。
- e. 如果您選擇建立新的部署，請選擇部署的目標核心裝置或物件群組。在指定目標頁面的部署目標下，選擇核心裝置或物件群組，然後選擇下一步。
- f. 在選取元件頁面上，確認已選取com.example.BatteryAwareHelloWorld元件，選擇下一步。
- g. 在設定元件頁面上，選取 com.example.BatteryAwareHelloWorld，然後執行下列動作：
 - i. 選擇設定元件。
 - ii. 在設定com.example.BatteryAwareHelloWorld模態、在組態更新下，在要合併的組態中，輸入下列組態更新。

```
{  
  "BatteryThreshold": 70  
}
```

- iii. 選擇確認以關閉模態，然後選擇下一步。
- h. 在確認進階設定頁面的部署政策區段的元件更新政策下，確認已選取通知元件。當您建立新的部署時，預設會選取通知元件。
- i. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署可能需要一分鐘的時間才能完成。

3. Core AWS IoT Greengrass 軟體會將 stdout 從元件程序儲存至 logs 資料夾中的日誌檔案。執行下列命令來驗證 Hello World 元件是否執行並列印狀態訊息。

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
```

您應該會看到類似下列範例的訊息。

```
Hello, World! Battery level (50) is below threshold (70), so the component will defer updates.
```

Note

如果檔案不存在，則部署可能尚未完成。如果檔案在 30 秒內不存在，則部署可能會失敗。例如，如果核心裝置沒有從 S3 儲存貯體下載元件成品的許可，就可能發生這種情況。執行下列命令以檢視 AWS IoT Greengrass Core 軟體日誌檔案。此檔案包含 Greengrass 核心裝置部署服務的日誌。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.log
```

type 命令會將檔案的內容寫入終端機。多次執行此命令，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

4. 建立新的核心裝置部署，以確認元件延遲更新。請執行下列操作：
 - a. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇部署。
 - b. 選擇您先前建立或修改的部署。
 - c. 在部署頁面上，選擇修訂。
 - d. 在修訂部署模式中，選擇修訂部署。
 - e. 在每個步驟選擇下一步，然後選擇部署。
5. 執行下列命令以再次檢視元件的日誌，並確認其延遲更新。

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
```

您應該會看到類似下列範例的訊息。元件會將更新延遲 30 秒，因此元件會重複列印此訊息。

```
Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
```

6. 使用文字編輯器編輯虛擬電池電量檔案，並將電池電量變更為超過閾值的值，以便部署可以繼續進行。
 - 在 Linux 核心裝置上，編輯名為的檔案/home/ggc_user/virtual_battery.json。執行具有 sudo 許可的文字編輯器。
 - 在 Windows 核心裝置上，編輯名為的檔案C:\Users\ggc_user\virtual_battery.json。以管理員身分執行文字編輯器。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
sudo nano /home/ggc_user/virtual_battery.json
```

將電池電量變更為 80。

```
{  
  "battery_level": 80  
}
```

7. 執行下列命令以再次檢視元件的日誌，並確認其確認更新。

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
```

您應該會看到類似下列範例的訊息。

```
Hello, World! Battery level (80) is above threshold (70), so the component will  
acknowledge updates.  
Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
```

您已完成本教學課程。Hello World 元件會根據核心裝置的電池電量延遲或認可更新。如需本教學課程探索主題的詳細資訊，請參閱以下內容：

- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#)
- [AWS IoT Greengrass 開發套件命令列介面](#)

教學課程：透過 MQTT 與本機 IoT 裝置互動

您可以完成本教學課程，將核心裝置設定為與透過 MQTT 連線至核心裝置之本機 IoT 裝置互動，稱為用戶端裝置。在本教學課程中，您將 AWS IoT 物件設定為使用雲端探索，以用戶端裝置的形式連線至核心裝置。當您設定雲端探索時，用戶端裝置可以將請求傳送至 AWS IoT Greengrass 雲端服務，以探索核心裝置。的回應 AWS IoT Greengrass 包含您設定用戶端裝置探索之核心裝置的連線資訊和憑證。然後，用戶端裝置可以使用此資訊連線到可用的核心裝置，以便透過 MQTT 進行通訊。

在此教學課程中，您將執行下列操作：

1. 視需要檢閱和更新核心裝置的許可。
2. 將用戶端裝置與核心裝置建立關聯，以便他們可以使用雲端探索來探索核心裝置。
3. 將 Greengrass 元件部署至核心裝置，以啟用用戶端裝置支援。
4. 將用戶端裝置連接至核心裝置，並測試與雲端服務的 AWS IoT Core 通訊。
5. 開發與用戶端裝置通訊的自訂 Greengrass 元件。
6. 開發與用戶端裝置影射 [AWS IoT 子](#) 互動的自訂元件。

本教學課程使用單一核心裝置和單一用戶端裝置。您也可以遵循教學課程來連接和測試多個用戶端裝置。

您可以預期在本教學課程中花費 30-60 分鐘。

先決條件

為了完成本教學，您需要以下項目：

- AWS 帳戶。如果您沒有帳戶，請參閱 [設定 AWS 帳戶](#)。
- 具有管理員許可的 AWS Identity and Access Management (IAM) 使用者。
- Greengrass 核心裝置。如需如何設定核心裝置的詳細資訊，請參閱 [設定 AWS IoT Greengrass 核心裝置](#)。
- 核心裝置必須執行 Greengrass 核 v2.6.0 或更新版本。此版本包含支援本機發佈/訂閱通訊中的萬用字元，以及支援用戶端裝置影子。

Note

用戶端裝置支援需要 Greengrass nucleus v2.2.0 或更新版本。不過，本教學課程會探索較新的功能，例如在本機發佈/訂閱中支援 MQTT 萬用字元，以及支援用戶端裝置影子。這些功能需要 Greengrass nucleus v2.6.0 或更新版本。

- 核心裝置必須與要連線的用戶端裝置位於相同的網路上。
- （選用）若要完成您開發自訂 Greengrass 元件的模組，核心裝置必須執行 Greengrass CLI。如需詳細資訊，請參閱 [安裝 Greengrass CLI](#)。
- 本教學課程中做為用戶端裝置連線的 AWS IoT 物件。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [建立 AWS IoT 資源](#)。
- 用戶端裝置的 AWS IoT 政策必須允許 `greengrass:Discover` 許可。如需詳細資訊，請參閱 [用戶端裝置的最低 AWS IoT 政策](#)。

- 用戶端裝置必須與核心裝置位於相同的網路上。
- 用戶端裝置必須執行 [Python 3](#)。
- 用戶端裝置必須執行 [Git](#)。

步驟 1：檢閱和更新核心裝置 AWS IoT 政策

若要支援用戶端裝置，核心裝置的 AWS IoT 政策必須允許下列許可：

- greengrass:PutCertificateAuthorities
- greengrass:VerifyClientDeviceIdentity
- greengrass:VerifyClientDeviceIoTCertificateAssociation
- greengrass:GetConnectivityInfo
- greengrass:UpdateConnectivityInfo – (選用) 需要此許可才能使用 [IP 偵測器元件](#)，該元件會將核心裝置的網路連線資訊報告給 AWS IoT Greengrass 雲端服務。

如需核心裝置的這些許可和 AWS IoT 政策的詳細資訊，請參閱 [AWS IoT 資料平面操作的 政策](#) 和 [支援用戶端裝置的最低 AWS IoT 政策](#)。

在本節中，您將檢閱核心裝置 AWS IoT 的政策，並新增任何缺少的必要許可。如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，您的核心裝置具有允許存取所有 AWS IoT Greengrass 動作 () AWS IoT 的政策 greengrass:*。在此情況下，只有在您計劃將影子管理員元件設定為與裝置影子同步時，才必須更新 AWS IoT 政策 AWS IoT Core。否則，您可以略過本節。

檢閱和更新核心裝置 AWS IoT 的政策

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇核心裝置。
2. 在核心裝置頁面上，選擇要更新的核心裝置。
3. 在核心裝置詳細資訊頁面上，選擇核心裝置物件的連結。此連結會在 主控台中 AWS IoT 開啟物件詳細資訊頁面。
4. 在物件詳細資訊頁面上，選擇憑證。
5. 在憑證索引標籤中，選擇物件的作用中憑證。
6. 在憑證詳細資訊頁面上，選擇政策。
7. 在政策索引標籤中，選擇要檢閱和更新 AWS IoT 的政策。您可以將必要的許可新增至連接到核心裝置作用中憑證的任何政策。

Note

如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，則有兩個 AWS IoT 政策。如果 GreengrassV2IoTThingPolicy 存在，我們建議您選擇名為 的政策。您使用快速安裝程式建立的核心裝置預設會使用此政策名稱。如果您將許可新增至此政策，您也會將這些許可授予使用此政策的其他核心裝置。

8. 在政策概觀中，選擇編輯作用中版本。
9. 檢閱必要許可的政策，並新增任何缺少的必要許可。
10. 若要將新的政策版本設定為作用中版本，請在政策版本狀態下，選取將編輯的版本設定為此政策的作用中版本。
11. 選擇儲存為新版本。

步驟 2：啟用用戶端裝置支援

若要讓用戶端裝置使用雲端探索連線到核心裝置，您必須建立裝置關聯。當您將用戶端裝置與核心裝置建立關聯時，您可以讓該用戶端裝置擷取核心裝置的 IP 地址和憑證，以用於連線。

若要讓用戶端裝置安全地連線至核心裝置，並與 Greengrass 元件通訊 AWS IoT Core，您可以將下列 Greengrass 元件部署至核心裝置：

- [用戶端裝置身分驗證](#) (`aws.greengrass.clientdevices.Auth`)

部署用戶端裝置身分驗證元件，以驗證用戶端裝置並授權用戶端裝置動作。此元件可讓您的 AWS IoT 物件連線至核心裝置。

此元件需要一些組態才能使用它。您必須指定用戶端裝置的群組，以及每個群組獲授權執行的操作，例如透過 MQTT 連線和通訊。如需詳細資訊，請參閱[用戶端裝置驗證元件組態](#)。

- [MQTT 3.1.1 代理程式 \(Moquette\)](#) (`aws.greengrass.clientdevices.mqtt.Moquette`)

部署 Moquette MQTT 代理程式元件以執行輕量型 MQTT 代理程式。Moquette MQTT 代理程式符合 MQTT 3.1.1，並包含 QoS 0、QoS 1、QoS 2、保留訊息、最後遺囑訊息和持久性訂閱的本機支援。

您不需要設定此元件即可使用它。不過，您可以設定此元件操作 MQTT 代理程式的連接埠。根據預設，它會使用連接埠 8883。

- [MQTT 橋接器](#) (`aws.greengrass.clientdevices.mqtt.Bridge`)

(選用) 部署 MQTT 橋接器元件，以在用戶端裝置 (本機 MQTT)、本機發佈/訂閱和 AWS IoT Core MQTT 之間轉送訊息。設定此元件以同步用戶端裝置與 Greengrass 元件中的用戶端裝置，AWS IoT Core 並與之互動。

此元件需要使用組態。您必須指定此元件轉送訊息的主題映射。如需詳細資訊，請參閱 [MQTT 橋接器元件組態](#)。

- [IP 偵測器](#) (aws.greengrass.clientdevices.IPDetector)

(選用) 部署 IP 偵測器元件，以自動向 AWS IoT Greengrass 雲端服務報告核心裝置的 MQTT 代理程式端點。如果您有複雜的網路設定，例如路由器將 MQTT 代理程式連接埠轉送至核心裝置的網路設定，則無法使用此元件。

您不需要設定此元件即可使用它。

在本節中，您可以使用 AWS IoT Greengrass 主控台來關聯用戶端裝置，並將用戶端裝置元件部署至核心裝置。

啟用用戶端裝置支援

1. 導覽至 [AWS IoT Greengrass 主控台](#)。
2. 在左側導覽功能表中，選擇核心裝置。
3. 在核心裝置頁面上，選擇您要啟用用戶端裝置支援的核心裝置。
4. 在核心裝置詳細資訊頁面上，選擇用戶端裝置索引標籤。
5. 在用戶端裝置索引標籤上，選擇設定雲端探索。

設定核心裝置探索頁面隨即開啟。在此頁面上，您可以將用戶端裝置與核心裝置建立關聯，並部署用戶端裝置元件。此頁面會在步驟 1：選取目標核心裝置中為您選取核心裝置。

Note

您也可以使用此頁面來設定物件群組的核心裝置探索。如果您選擇此選項，您可以將用戶端裝置元件部署到物件群組中的所有核心裝置。不過，如果您選擇此選項，則必須在建立部署之後，手動將用戶端裝置與每個核心裝置建立關聯。在本教學課程中，您會設定單一核心裝置。

6. 在步驟 2：關聯用戶端裝置中，將用戶端裝置的 AWS IoT 物件與核心裝置建立關聯。這可讓用戶端裝置使用雲端探索來擷取核心裝置的連線資訊和憑證。請執行下列操作：

- a. 選擇關聯用戶端裝置。
 - b. 在將用戶端裝置與核心裝置模型建立關聯中，輸入要建立關聯的 AWS IoT 物件名稱。
 - c. 選擇新增。
 - d. 選擇關聯。
7. 在步驟 3：設定和部署 Greengrass 元件中，部署元件以啟用用戶端裝置支援。如果目標核心裝置具有先前的部署，此頁面會修訂該部署。否則，此頁面會為核心裝置建立新的部署。執行下列動作來設定和部署用戶端裝置元件：
- a. 核心裝置必須執行 [Greengrass 核 v2.6.0](#) 或更新版本，才能完成本教學課程。如果核心裝置執行舊版，請執行下列動作：
 - i. 選取方塊以部署aws.greengrass.Nucleus元件。
 - ii. 針對aws.greengrass.Nucleus元件，選擇編輯組態。
 - iii. 針對元件版本，選擇 2.6.0 版或更新版本。
 - iv. 選擇確認。

 Note

如果您從較早的次要版本升級 Greengrass 核，且核心裝置執行 提供的元件，而這些[AWS元件](#)相依於核，則您也必須將 AWS提供的元件更新為較新的版本。您可以在本教學稍後檢閱部署時設定這些元件的版本。如需詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

- b. 針對aws.greengrass.clientdevices.Auth元件，選擇編輯組態。
- c. 在編輯用戶端裝置身分驗證元件的組態模式中，設定授權政策，允許用戶端裝置在核心裝置上發佈和訂閱 MQTT 代理程式。請執行下列操作：
 - i. 在組態下，在要合併程式碼的組態區塊中，輸入下列組態，其中包含用戶端裝置授權政策。每個裝置群組授權政策都會指定一組動作，以及用戶端裝置可以對其執行這些動作的資源。
 - 此政策允許名稱開頭為 的用戶端裝置MyClientDevice，在所有 MQTT 主題上進行連線和通訊。將 *MyClientDevice** 取代為做為用戶端裝置連線的 AWS IoT 物件名稱。您也可以使用符合用戶端裝置名稱的*萬用字元指定名稱。* 萬用字元必須位於名稱的結尾。

如果您有第二個用戶端裝置要連線，請將 *MyOtherClientDevice** 取代為該用戶端裝置的名稱，或是與該用戶端裝置名稱相符的萬用字元模式。否則，您可以移除或保留選取規則的此區段，允許名稱相符的用戶端裝置 *MyOtherClientDevice** 進行連線和通訊。

- 此政策使用 OR 運算子，也允許名稱開頭為 的用戶端裝置 *MyOtherClientDevice* 在所有 MQTT 主題上連線和通訊。您可以在選取規則中移除此子句，或修改它以符合要連線的用戶端裝置。
- 此政策允許用戶端裝置在所有 MQTT 主題上發佈和訂閱。若要遵循最佳安全實務，請將 `mqtt:publish` 和 `mqtt:subscribe` 操作限制在用戶端裝置用於通訊的最低限度主題集。

```
{
  "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": [
    "*"
  ]
}
}
}
}
```

如需詳細資訊，請參閱[用戶端裝置驗證元件組態](#)。

- ii. 選擇確認。
- d. 針對aws.greengrass.clientdevices.mqtt.Bridge元件，選擇編輯組態。
- e. 在編輯 MQTT 橋接器元件的組態模態中，設定主題映射，將 MQTT 訊息從用戶端裝置轉送至 AWS IoT Core。請執行下列操作：
 - i. 在組態下，於要合併程式碼的組態區塊中，輸入下列組態。此組態指定 將clients/+/
hello/world主題篩選條件上的 MQTT 訊息從用戶端裝置轉送至 AWS IoT Core 雲端服務。例如，此主題篩選條件符合clients/MyClientDevice1/hello/world主題。

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

如需詳細資訊，請參閱 [MQTT 橋接器元件組態](#)。

- ii. 選擇確認。

8. 選擇檢閱並部署，以檢閱此頁面為您建立的部署。
9. 如果您先前尚未在此區域中設定 [Greengrass 服務角色](#)，主控台會開啟模式來為您設定服務角色。用戶端裝置身分驗證元件會使用此服務角色來驗證用戶端裝置的身分，而 IP 偵測器元件會使用此服務角色來管理核心裝置連線資訊。選擇 授予許可。
10. 在檢閱頁面上，選擇部署以開始部署至核心裝置。
11. 若要驗證部署是否成功，請檢查部署的狀態，並檢查核心裝置上的日誌。若要檢查核心裝置上的部署狀態，您可以在部署概觀中選擇目標。如需詳細資訊，請參閱下列內容：
 - [檢查部署狀態](#)
 - [監控 AWS IoT Greengrass 日誌](#)

步驟 3：連接用戶端裝置

用戶端裝置可以使用 AWS IoT Device SDK 來探索、連線核心裝置，以及與核心裝置通訊。用戶端裝置必須是 AWS IoT 物件。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[建立物件](#)。

在本節中，您將安裝 [AWS IoT Device SDK v2 for Python](#)，並從執行 Greengrass 探索範例應用程式 AWS IoT Device SDK。

Note

AWS IoT Device SDK 也提供其他程式設計語言的。本教學課程使用適用於 Python 的 AWS IoT Device SDK v2，但您可以針對您的使用案例探索其他 SDKs。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南[AWS IoT](#)》中的 [裝置SDKs](#)。

將用戶端裝置連接到核心裝置

1. 將 [AWS IoT Device SDK v2 for Python](#) 下載並安裝到 AWS IoT 物件，以做為用戶端裝置進行連線。

在用戶端裝置上，執行下列動作：

- a. 複製 AWS IoT Device SDK v2 for Python 儲存庫以進行下載。

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

- b. 安裝 AWS IoT Device SDK v2 for Python。

```
python3 -m pip install --user ./aws-iot-device-sdk-python-v2
```

2. 變更為適用於 Python 的 AWS IoT Device SDK v2 中的範例資料夾。

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

3. 執行範例 Greengrass 探索應用程式。此應用程式需要指定用戶端裝置物件名稱的引數、要使用的 MQTT 主題和訊息，以及驗證和保護連線的憑證。下列範例會將 Hello World 訊息傳送至 `clients/MyClientDevice1/hello/world` 主題。

 Note

本主題與您設定 MQTT 橋接器以將訊息轉送至 AWS IoT Core 先前主題相符。

- 將 `MyClientDevice1` 取代為用戶端裝置的物件名稱。
- 將 `~/certs/AmazonRootCA1.pem` 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
- 將 `~/certs/device.pem.crt` 取代為用戶端裝置上裝置憑證的路徑。
- 將 `~/certs/private.pem.key` 取代為用戶端裝置上私有金鑰檔案的路徑。
- 將 `us-east-1` 取代為用戶端裝置和核心裝置操作所在的 AWS 區域。

```
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
```

探索範例應用程式會傳送訊息 10 次並中斷連線。它也會訂閱發佈訊息的相同主題。如果輸出指出應用程式收到有關主題的 MQTT 訊息，用戶端裝置可以成功與核心裝置通訊。

```
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}'

```

如果應用程式輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

您也可以核心裝置上檢視 Greengrass 日誌，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

4. 確認 MQTT 橋接器將訊息從用戶端裝置轉送至 AWS IoT Core。您可以在 AWS IoT Core 主控台中使用 MQTT 測試用戶端來訂閱 MQTT 主題篩選條件。請執行下列操作：
 - a. 導覽至 [AWS IoT 主控台](#)。
 - b. 在左側導覽選單的測試下，選擇 MQTT 測試用戶端。
 - c. 在訂閱主題索引標籤上，針對主題篩選條件，輸入 `clients/+ /hello/world` 以從核心裝置訂閱用戶端裝置訊息。

- d. 選擇 Subscribe (訂閱)。
- e. 再次在用戶端裝置上執行發佈/訂閱應用程式。

MQTT 測試用戶端會在符合此主題篩選條件的主題上顯示您從用戶端裝置傳送的訊息。

步驟 4：開發與用戶端裝置通訊的元件

您可以開發與用戶端裝置通訊的 Greengrass 元件。元件使用[程序間通訊 \(IPC\)](#) 和本機發佈/訂閱界面，在核心裝置上進行通訊。若要與用戶端裝置互動，請設定 MQTT 橋接器元件，以在用戶端裝置與本機發佈/訂閱界面之間轉送訊息。

在本節中，您會更新 MQTT 橋接器元件，將訊息從用戶端裝置轉送至本機發佈/訂閱界面。然後，您會開發訂閱這些訊息的元件，並在收到訊息時列印訊息。

開發與用戶端裝置通訊的元件

1. 修訂核心裝置的部署，並設定 MQTT 橋接器元件，將訊息從用戶端裝置轉送至本機發佈/訂閱。請執行下列操作：
 - a. 導覽至 [AWS IoT Greengrass 主控台](#)。
 - b. 在左側導覽功能表中，選擇核心裝置。
 - c. 在核心裝置頁面上，選擇您用於本教學課程的核心裝置。
 - d. 在核心裝置詳細資訊頁面上，選擇用戶端裝置索引標籤。
 - e. 在用戶端裝置索引標籤上，選擇設定雲端探索。

設定核心裝置探索頁面隨即開啟。在此頁面上，您可以變更或設定部署到核心裝置的用戶端裝置元件。

- f. 在步驟 3 中，針對aws.greengrass.clientdevices.mqtt.Bridge元件選擇編輯組態。
- g. 在編輯 MQTT 橋接器元件的組態模式中，設定主題映射，將 MQTT 訊息從用戶端裝置轉送至本機發佈/訂閱界面。請執行下列操作：
 - i. 在組態下，於要合併程式碼的組態區塊中，輸入下列組態。此組態指定 將主題上的 MQTT 訊息從用戶端裝置符合clients/+ /hello/world主題篩選條件，轉送至 AWS IoT Core 雲端服務和本機 Greengrass 發佈/訂閱代理程式。

```
{
  "mqttTopicMapping": {
    "HelloWorldIoTCoreMapping": {
```

```
    "topic": "clients/+/hello/world",
    "source": "LocalMqtt",
    "target": "IotCore"
  },
  "HelloWorldPubsubMapping": {
    "topic": "clients/+/hello/world",
    "source": "LocalMqtt",
    "target": "Pubsub"
  }
}
```

如需詳細資訊，請參閱 [MQTT 橋接器元件組態](#)。

ii. 選擇確認。

h. 選擇檢閱並部署，以檢閱此頁面為您建立的部署。

i. 在檢閱頁面上，選擇部署以開始部署至核心裝置。

j. 若要驗證部署是否成功，請檢查部署的狀態，並檢查核心裝置上的日誌。若要檢查核心裝置上的部署狀態，您可以在部署概觀中選擇目標。如需詳細資訊，請參閱下列內容：

- [檢查部署狀態](#)
- [監控 AWS IoT Greengrass 日誌](#)

2. 開發和部署從用戶端裝置訂閱 Hello World 訊息的 Greengrass 元件。請執行下列操作：

a. 在核心裝置上建立配方和成品的資料夾。

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

您必須針對成品資料夾路徑使用下列格式。包含您在配方中指定的元件名稱和版本。

```
artifacts/componentName/componentVersion/
```

- b. 使用文字編輯器建立具有下列內容的元件配方。此配方指定安裝適用於 Python 的 AWS IoT Device SDK v2，並執行訂閱主題並列印訊息的指令碼。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
```

將下列配方複製到 檔案。

```
{
  "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. 使用文字編輯器，以下列內容建立名為 `hello_world_subscriber.py` 的 Python 指令碼成品。此應用程式使用發佈/訂閱 IPC 服務來訂閱主題 `clients/+/hello/world` 並列印收到的訊息。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

將下列 Python 程式碼複製到 檔案。

```

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

此元件使用適用於 Python 的 V2 中的 IPC 用戶端 V2 與 AWS IoT Greengrass 核心軟體通訊。[AWS IoT Device SDK](#)相較於原始 IPC 用戶端，IPC 用戶端 V2 可減少您在自訂元件中使用 IPC 所需的程式碼量。

- d. 使用 Greengrass CLI 部署元件。

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. 檢視元件日誌，確認元件安裝成功並訂閱主題。

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
```

您可以保持日誌饋送開啟，以驗證核心裝置是否接收訊息。

4. 在用戶端裝置上，再次執行範例 Greengrass 探索應用程式，以傳送訊息至核心裝置。

```
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. 再次檢視元件日誌，以確認元件從用戶端裝置接收和列印訊息。

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
```

步驟 5：開發與用戶端裝置陰影互動的元件

您可以開發與用戶端裝置[AWS IoT 影子](#)互動的 Greengrass 元件。影子是 JSON 文件，可存放 AWS IoT 物件的目前或所需狀態資訊，例如用戶端裝置。即使用戶端裝置未連線，自訂元件也可以存取用戶端裝置的影子來管理其狀態 AWS IoT。每個 AWS IoT 物件都有一個未命名的影子，您也可以為每個物件建立多個命名的影子。

在本節中，您會部署[影子管理員元件](#)來管理核心裝置上的影子。您也可以更新 MQTT 橋接器元件，在用戶端裝置和影子管理員元件之間轉送影子訊息。然後，您會開發更新用戶端裝置影子的元件，並在用戶端裝置上執行範例應用程式，以回應元件的影子更新。此元件代表智慧光管理應用程式，其中核心裝置會管理以用戶端裝置身分連接到它的智慧光的顏色狀態。

開發與用戶端裝置影子互動的元件

1. 修改核心裝置的部署，以部署影子管理員元件，並設定 MQTT 橋接器元件，在用戶端裝置與本機發佈/訂閱之間轉送影子訊息，影子管理員會在其中通訊。請執行下列操作：
 - a. 導覽至 [AWS IoT Greengrass 主控台](#)。
 - b. 在左側導覽功能表中，選擇核心裝置。
 - c. 在核心裝置頁面上，選擇您用於本教學課程的核心裝置。

- d. 在核心裝置詳細資訊頁面上，選擇用戶端裝置索引標籤。
- e. 在用戶端裝置索引標籤上，選擇設定雲端探索。

設定核心裝置探索頁面隨即開啟。在此頁面上，您可以變更或設定部署到核心裝置的用戶端裝置元件。

- f. 在步驟 3 中，針對aws.greengrass.clientdevices.mqtt.Bridge元件選擇編輯組態。
- g. 在編輯 MQTT 橋接器元件的組態模式中，設定主題映射，在用戶端裝置與本機發佈/訂閱界面之間轉送[裝置影子主題](#)上的 MQTT 訊息。您也可以確認部署指定相容的 MQTT 橋接器版本。用戶端裝置陰影支援需要 MQTT 橋接器 v2.2.0 或更新版本。請執行下列操作：
 - i. 針對元件版本，選擇 2.2.0 版或更新版本。
 - ii. 在組態下，於要合併程式碼區塊的組態中，輸入下列組態。此組態指定 轉送影子主題的 MQTT 訊息。

```
{
  "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"
    }
  }
}
```

如需詳細資訊，請參閱 [MQTT 橋接器元件組態](#)。

- iii. 選擇確認。
 - h. 在步驟 3 中，選取要部署的aws.greengrass.ShadowManager元件。
 - i. 選擇檢閱並部署，以檢閱此頁面為您建立的部署。
 - j. 在檢閱頁面上，選擇部署以開始部署至核心裝置。
 - k. 若要驗證部署是否成功，請檢查部署的狀態，並檢查核心裝置上的日誌。若要檢查核心裝置上的部署狀態，您可以在部署概觀中選擇目標。如需詳細資訊，請參閱下列內容：
 - [檢查部署狀態](#)
 - [監控 AWS IoT Greengrass 日誌](#)
2. 開發和部署管理智慧光線用戶端裝置的 Greengrass 元件。請執行下列操作：
- a. 在核心裝置上建立元件成品的資料夾。

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

您必須針對成品資料夾路徑使用下列格式。包含您在配方中指定的元件名稱和版本。

```
artifacts/componentName/componentVersion/
```

- b. 使用文字編輯器建立具有下列內容的元件配方。此配方指定安裝適用於 Python 的 AWS IoT Device SDK v2，並執行與智慧光線用戶端裝置的影子互動的指令碼，以管理其顏色。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
```

將下列配方複製到 檔案。

```
{
  "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 awsiot-sdk",
            "Run": "python3 -u {artifacts:path}/smart_light_manager.py"
        }
    },
    {
        "Platform": {
            "os": "windows"
        },
        "Lifecycle": {
            "install": "py -3 -m pip install --user awsiot-sdk",
            "Run": "py -3 -u {artifacts:path}/smart_light_manager.py"
        }
    }
]
}

```

- c. 使用文字編輯器，以下列內容建立名為 `smart_light_manager.py` 的 Python 指令碼成品。此應用程式使用影子 IPC 服務來取得和更新用戶端裝置影子，以及本機發佈/訂閱 IPC 服務來接收報告的影子更新。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```

nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py

```

將下列 Python 程式碼複製到 檔案。

```
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()

```

此 Python 應用程式會執行下列動作：

- 讀取元件的組態，以取得要管理的智慧光源用戶端裝置清單。
 - 使用 [SubscribeToConfigurationUpdate](#) IPC 操作訂閱組態更新通知。AWS IoT Greengrass 核心軟體會在每次元件的組態變更時傳送通知。當元件收到組態更新通知時，它會更新其管理的智慧光源用戶端裝置清單。
 - 取得每個智慧光用戶端裝置的影子，以取得其初始顏色狀態。
 - 將每個智慧光用戶端裝置的顏色設定為每 15 秒隨機的顏色。元件會更新用戶端裝置的物件影子，以變更其顏色。此操作會透過 MQTT 將陰影差異事件傳送至用戶端裝置。
 - 使用 [SubscribeToTopic](#) IPC 操作，在本機發佈/訂閱界面上訂閱陰影更新接受的訊息。此元件會收到這些訊息，以追蹤每個智慧光用戶端裝置的顏色。當智慧光用戶端裝置收到陰影更新時，會傳送 MQTT 訊息以確認它已收到更新。MQTT 橋接器會將此訊息轉送至本機發佈/訂閱界面。
- d. 使用 Greengrass CLI 部署元件。部署此元件時，您可以指定用戶端裝置的清單 `smartLightDeviceNames`，其會管理其影子。將 *MyClientDevice1* 取代為用戶端裝置的物件名稱。

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. 檢視元件日誌，以驗證元件是否安裝並成功執行。

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
```

元件會傳送變更智慧光用戶端裝置顏色的請求。影子管理員會收到請求並設定影子desired的狀態。不過，智慧光用戶端裝置尚未執行，因此陰影reported的狀態不會變更。元件的日誌包含下列訊息。

```
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}
```

您可以保持日誌饋送開啟，以查看元件何時列印訊息。

4. 下載並執行使用 Greengrass 探索並訂閱裝置陰影更新的範例應用程式。在用戶端裝置上，執行下列動作：
 - a. 使用文字編輯器建立名為 `basic_discovery_shadow.py` 的 Python 指令碼，其中包含下列內容。此應用程式使用 Greengrass 探索和陰影，在用戶端裝置和核心裝置之間保持屬性同步。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano basic_discovery_shadow.py
```

將下列 Python 程式碼複製到檔案。

```
# 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_interrupted(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_interrupted,
                    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()
```

此 Python 應用程式會執行下列動作：

- 使用 Greengrass 探索來探索和連線至核心裝置。
- 從核心裝置請求影子文件，以取得屬性的初始狀態。
- 訂閱至影子差異事件，核心裝置會在屬性desired的值與其reported值不同時傳送。當應用程式收到陰影差異事件時，它會變更 屬性的值，並將更新傳送至核心裝置，以將新值設定為其reported值。

此應用程式結合了來自 v2 的 AWS IoT Device SDK Greengrass 探索和陰影範例。

- b. 執行範例應用程式。此應用程式需要指定用戶端裝置物件名稱的引數、要使用的影子屬性，以及驗證和保護連線的憑證。
 - 將 *MyClientDevice1* 取代為用戶端裝置的物件名稱。
 - 將 *~/certs/AmazonRootCA1.pem* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
 - 將 *~/certs/device.pem.crt* 取代為用戶端裝置上裝置憑證的路徑。
 - 將 *~/certs/private.pem.key* 取代為用戶端裝置上私有金鑰檔案的路徑。
 - 將 *us-east-1* 取代為用戶端裝置和核心裝置操作所在的 AWS 區域。


```
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
```

範例應用程式會訂閱影子主題，並等待接收來自核心裝置的影子差異事件。如果輸出指出應用程式接收並回應影子差異事件，用戶端裝置可以成功與其核心裝置上的影子互動。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GG
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.
```

如果應用程式輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

您也可以在核心裝置上檢視 Greengrass 日誌，以驗證用戶端裝置是否成功連線並傳送訊息。

如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

5. 再次檢視元件日誌，以驗證元件是否收到來自智慧光源用戶端裝置的陰影更新確認。

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
```

元件會記錄訊息，以確認智慧光用戶端裝置已變更其顏色。

```
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

用戶端裝置的影子在核心裝置和用戶端裝置之間同步。不過，核心裝置不會與用戶端裝置的影子同步 AWS IoT Core。例如，您可以將影子與 同步 AWS IoT Core，以檢視或修改機群中所有裝置的狀態。如需如何設定影子管理員元件以與 同步影子的詳細資訊 AWS IoT Core，請參閱 [使用 同步本機裝置陰影 AWS IoT Core](#)。

您已完成本教學課程。用戶端裝置會連線至核心裝置，將 MQTT 訊息傳送至 AWS IoT Core 和 Greengrass 元件，並從核心裝置接收陰影更新。如需本教學課程涵蓋主題的詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)
- [測試用戶端裝置通訊](#)
- [Greengrass 探索 RESTful API](#)
- [在用戶端裝置與 之間轉送 MQTT 訊息 AWS IoT Core](#)
- [與元件中的用戶端裝置互動](#)
- [與裝置陰影互動](#)
- [與用戶端裝置影子互動和同步](#)

教學課程：使用信任平台模組 (TPM) 保護 Greengrass Nucleus

Note

本教學課程中的機制僅支援 [使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

這些教學課程包含有關如何使用 TPM2 晶片作為硬體安全模組 (HSM) 來建立私有金鑰和 CSR 的說明。用於 [建立物件憑證](#)。

這些教學課程說明如何使用 PKCS#11 介面，透過信任平台模組 (TPM) 設定 AWS IoT Greengrass Core 軟體 (Greengrass Nucleus) 來增強裝置安全性。此 TPM 整合可確保用於裝置身分和連線至 的私有金鑰和憑證 AWS IoT Core 安全地存放在防竄改硬體中，防止擷取模擬或其他惡意活動。

當您完成此整合時，您的 Greengrass 核心裝置將使用 TPM 保護的私有金鑰來進行身分和 AWS IoT 與服務的通訊。

如需 Greengrass 裝置安全性的詳細資訊，請參閱 [中的安全性 AWS IoT Greengrass](#)。

先決條件

為了完成本教學，您需要以下項目：

- 具有 TPM 2.0 硬體或韌體 TPM 2.0 的 Linux 相容裝置。

- 本教學中的指示是針對 Ubuntu 24.04 LTS 定義。
 - 任何可支援 Linux [TPM2 軟體堆疊的 Linux](#) 發行版本都可以支援此機制。
- 安裝 AWS CLI 並設定 的開發人員機器具有以下許可：
 - 建立和管理 AWS IoT 資源
 - 建立和管理 IAM 角色和政策
- 安裝在您裝置上的 Java 執行期環境 (JRE) 第 8 版或更新版本。
- 您的裝置上安裝了下列軟體套件：
 - curl
 - jq
- 裝置上的根或 sudo 權限。

步驟 1：安裝 TPM2 工具和相依性

在此步驟中，您會安裝必要的 TPM2 軟體工具和程式庫。

1. 執行下列命令，以更新您的套件管理員並安裝 TPM2 工具和相依性。

```
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. 在使用 OpenSSL 引擎 3 的 Ubuntu 24.04 上安裝 OpenSSL TPM2 提供者套件。

```
sudo apt-get install tpm2-openssl
```

步驟 2：初始化 PKCS#11 存放區並建立槽

1. 建立目錄以存放資料。

```
sudo mkdir -p /etc/tpm2_pkcs11
```

2. 將儲存位置設定為環境變數。如需存放階層的詳細資訊，請參閱[初始化](#)。

```
export TPM2_PKCS11_STORE=/etc/tpm2_store
```

3. 使用主要物件初始化 TPM2 字符。

```
sudo tpm2_ptool init
```

可用的選項如下：

`hierarchy-auth HIERARCHY_AUTH`

將主要物件新增至階層的授權密碼。

`primary-auth PRIMARY_AUTH`

現有主索引鍵物件的授權值。

預設是空的身分驗證值。

`primary-handle [PRIMARY_HANDLE]`

使用現有的主索引鍵物件。

預設：0x81000001

`transient-parents`

使用指定範本的暫時性主要物件。

值：tpm2-tools-default、tpm2-tools-ecc-default、tss2-engine-key

path PATH

存放區目錄的位置。如果指定，目錄必須存在。如果未指定，它會查看環境變數來執行搜尋TPM2_PKCS11_STORE。如果未設定該環境變數，則會查看 /etc/tpm2_pkcs11。如果找不到或無法建立該目錄，則會預設為目前的工作目錄。

步驟 3：建立權杖和金鑰

1. 建立 PKCS#11 權杖。

```
sudo tpm2_ptool addtoken --pid=1 --userpin=USERPIN --sopin=SOPIN --label=greengrass
```

可用的選項如下：

--pid PID

要與此字符建立關聯的主要物件 ID。

--sopin SOPIN

管理員 PIN 碼。此接腳用於物件復原。

--userpin USERPIN

使用者 PIN 碼。此 PIN 用於物件使用的身分驗證。

--label LABEL

用於識別使用中設定檔的唯一標籤必須是唯一的。

--hierarchy-auth HIERARCHY_AUTH

2. 建立 ECC 金鑰物件。

```
sudo tpm2_ptool addkey --algorithm=ecc256 --label=greengrass --userpin=***** --key-label=greenkey
```

`--label LABEL`

要匯入金鑰的字符標籤。

`--key-label KEY_LABEL`

匯入的金鑰標籤。預設為整數值。

`--id ID`

金鑰 ID。預設為隨機 8 個位元組的十六進位。

`--attr-always-authenticate`

將 `CKA_ALWAYS_AUTHENTICATE` 屬性設定為 `CK_TRUE`。

`--hierarchy-auth HIERARCHY_AUTH`

暫時物件所需的階層驗證。

`--sopin SOPIN`

管理員 PIN 碼。

`--userpin USERPIN`

使用者 PIN 碼。

`--algorithm`

{rsa1024 , rsa2048 , rsa3072 , rsa4096 , aes128 , aes256 , ecc224 , ecc256 , ecc384 , ecc521 , hmac : sha1 , hmac : sha256 , hmac : sha384 , hmac : sha512}

金鑰的類型。

3. 從權杖匯出 TPM2-TSS 物件以擷取身分驗證資料。

```
yaml_ecc0=$(sudo tpm2_ptool export --label="greengrass" --key-label="greenkey" --
userpin="*****")
```

輸出範例：

```
> echo $yaml_ecc0
```

```
object-auth: 706c1cad8a5238871b30149705255926
primary-object:
  auth: ''
  hierarchy: owner
  is_transient: false
```

Note

您也可以在執行此命令的目錄中找到名為 `greenkey.pem` 的檔案，這是 TSS2 私有金鑰。使用此選項來產生具有 tpm2 openssl 提供者的 CSR。TSS2 PRIVATE KEY 檔案受到 TPM 保護，無法在其他機器上使用。如需使用 OpenSSL 的 TSS2 金鑰詳細資訊，請參閱[儲存私有或公有金鑰](#)。

4. 擷取 TSS 私有金鑰的身分驗證資料。

```
auth_ecc0=$(echo "$yaml_ecc0" | grep "object-auth" | cut -d' ' -f2-)
```

步驟 4：產生憑證簽署請求 (CSR)

在此步驟中，您將使用 TPM2-protected 私有金鑰來產生 CSR。

1. 使用 TPM2 提供者產生 CSR。

```
sudo openssl req -new -provider tpm2 -provider base -key greenkey.pem -passin "pass:$auth_ecc0" -out "$H"$HOSTNAME".csr"
```

出現提示時，請提供 CSR 的必要資訊，包括：

- 國家名稱 (2 個字母代碼)
- State or Province Name (州或省名稱)
- Locality Name (地區名稱)
- Organization Name (組織名稱)
- Organizational Unit Name (組織單位名稱)
- Common Name (通用名稱)

- 電子郵件地址

或者，您可以為自動產生提供 OpenSSL 組態檔案。如需詳細資訊，請參閱 [OpenSSL 文件](#)。

2. 如果您未在相同機器上產生 CSR，請將產生的 CSR 複製到已設定 AWS 登入資料的機器。

步驟 5：建立物件憑證

建立 AWS IoT 物件憑證。如需如何建立物件憑證的詳細資訊，請參閱 [建立物件憑證](#)。

步驟 6：將物件憑證匯入 TPM

1. 將物件憑證複製到裝置。
2. 將物件憑證新增至 Greengrass 字符合。

```
sudo tpm2_ptool addcert --label=greengrass --key-label=greenkey device.pem.crt
```

可用的選項如下：

`--help`

顯示此說明訊息並結束。

`--label LABEL`

要移除的設定檔標籤。

`--key-label KEY_LABEL`

相關聯的私有金鑰標籤。

`--key-id KEY_ID`

十六進位中的關聯私有金鑰 ID。

`cert`

要新增的 x509 PEM 憑證。

步驟 7：擷取 PKCS#11 物件 URL

我們將使用 `gnutls-bin` 套件 `p11tool` 提供的 來取得 PKCS#11 字符合 URL 和物件 URLs。

1. 擷取 Greengrass 字符的字符 URL。

```
TOKEN=sudo p11tool --list-token-urls | grep "token=greengrass"
```

2. 取得 Greengrass 權杖的物件 URLs。使用您在步驟 3 中使用的相同接腳。

```
sudo p11tool --login --list-all "${TOKEN}"
```

輸出範例：

```
Token 'greengrass' with URL
'pkcs11:model=SLB9672%00%00%00%00%00%00%00%00;manufacturer=Infineon;serial=0000000000000000
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=0000000000000000
%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=0000000000000000
%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=0000000000000000
%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. 擷取私有金鑰和憑證的物件 URL。

步驟 8：使用 TPM2 支援設定和安裝 Greengrass

1. 設定物件憑證。如需詳細資訊，[請參閱設定物件憑證](#)。
2. 完成在的 HSM 中使用私有金鑰和憑證安裝 AWS IoT Greengrass 核心軟體的指示[安裝 AWS IoT Greengrass 核心軟體](#)。然後，使用下列步驟來設定您的安裝，透過 PKCS#11 介面利用 TPM2。
3. 確認您已將 PKCS#11 提供者元件下載並儲存在 Greengrass 安裝程式位置。
4. 使用文字編輯器來建立名為的組態檔案 config.yaml，以提供給安裝程式。例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 建立檔案。

```
nano GreengrassInstaller/config.yaml
```

5. 將下列 YAML 內容複製到檔案。此部分組態檔案會指定系統參數、Greengrass 核參數和 PKCS#11 提供者參數。

```
---
system:
  certificateFilePath: "pkcs11:model=SW%20%20%20TPM
%00%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%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. 使用適用於您安裝的參數編輯 檔案。

- a. 使用步驟 7 中擷取的 PKCS#11 URL 更新 certificateFilePath privateKeyPath 和 privateKeyPath，並使用更新 certificateFilePath 和 privateKeyPath。
- b. 根據您的 AWS IoT 端點更新您的 iotDataEndpoint 和 iotCredEndpoint。
- c. 在aws.greengrass.crypto.Pkcs11Provider組態中，根據您的平台更新程式庫。

Note

顯示的範例適用於 X86_64。ARM64 裝置的檔案路徑將相似。

7. 完成 中的 Greengrass 安裝步驟[安裝 AWS IoT Greengrass 核心軟體](#)。

步驟 9：驗證安裝

在此步驟中，您將驗證 Greengrass 透過 TPM2 整合正常執行。

1. 檢查 Greengrass 服務狀態。

```
sudo systemctl status greengrass.service
```

2. 檢視 Greengrass 日誌以確保沒有錯誤。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

3. 確認您的裝置在[AWS IoT 主控台](#)中顯示為已連線。

- a. 登入 [AWS IoT Greengrass 主控台](#)。
- b. 在管理下，展開 Greengrass 裝置，然後選擇核心裝置。
- c. 確認您的裝置已連線。裝置狀態HEALTHY會顯示是否已連線。如需詳細資訊，請參閱[檢查 Greengrass 核心裝置狀態](#)。

疑難排解

如果您在設定或操作TPM2-enabled 的 Greengrass 裝置期間遇到問題，請嘗試下列疑難排解步驟。

- 檢查主要 Greengrass 日誌檔案。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

- 驗證 PKCS#11 提供者組態。

```
sudo cat /greengrass/v2/config/effectiveConfig.yaml
```

- 確保 TPM2 服務正在執行。

```
sudo systemctl status tpm2-abrmd.service
```

- 確認 TPM2 金鑰可供存取。

```
sudo pkcs11-tool -module /usr/lib/x86_64-linux-gnu/pkcs11/libtpm2_pkcs11.so -l -p  
123456 -list-objects
```

- 如果您的作業系統設定為使用 TPM2 儲存根金鑰的完整磁碟加密，例如 Clevis 或 systemd-cryptenroll，請確認您使用的持久性控制代碼與這些工具所使用的相同。使用相同的持久性控制代碼可能會影響您的磁碟加密機制。若要檢查所有已建立和使用的持久性控點，請執行下列命令

```
sudo tpm2_getcap handles-persistent
```

後續步驟

現在您已成功將 Greengrass 核心裝置與 TPM2 整合，您可以：

- 將元件部署到您的安全 Greengrass 裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)
- 使用 TPM2 整合設定額外的 Greengrass 裝置。

如需 Greengrass 裝置安全性的詳細資訊，請參閱[中的安全性 AWS IoT Greengrass](#)。

教學課程：使用信任平台模組 (TPM) 保護 AWS IoT Greengrass Nucleus Lite

本教學課程說明如何啟用和設定對 AWS IoT Greengrass nucleus lite 的信任平台模組 (TPM) 支援。TPM 為安全金鑰儲存提供硬體型信任根。此安全功能可保護密碼編譯操作和敏感登入資料，增強裝置的安全性和完整性。

當您完成此整合時，您的 AWS IoT Greengrass 核心裝置將使用 TPM 保護的私有金鑰來進行身分和 AWS IoT 與服務的通訊。

如需 AWS IoT Greengrass 裝置安全性的詳細資訊，請參閱 [中的安全性 AWS IoT Greengrass](#)。

Important

只有使用手動資源佈建安裝 AWS IoT Greengrass nucleus lite 時，才支援此機制。

先決條件

為了完成本教學，您需要以下項目：

- 具有 TPM 2.0 硬體或 [NitroTPM](#) 的 Linux 相容裝置
- 安裝 Greengrass Nucleus Lite 的開發人員機器。如需詳細資訊，請參閱[安裝 AWS IoT Greengrass 核心軟體（主控台）](#)。
- 本教學中的指示是針對 Ubuntu 24.04 LTS 定義。
- 任何可支援 Linux [TPM2 軟體堆疊的 Linux](#) 發行版本都可以支援此機制。
- [AWS CLI](#) 已安裝並設定具有下列許可的開發人員機器：
 - 建立和管理 AWS IoT 資源
 - 建立和管理 IAM 角色和政策
- 裝置上的根或 sudo 權限。

本教學課程包含如何使用 TPM2 晶片做為硬體安全模組 (HSM) 來建立私有金鑰和用於建立 AWS IoT 物件憑證的 CSR 的說明。

步驟 1：設定 NitroTPM 執行個體

1. 設定 NitroTPM 執行個體。如需詳細資訊，請參閱 [NitroTPM](#)。
2. 使用上一個步驟中建立的自訂 AMI 啟動最終執行個體。

Important

與 SSH 連線時，請使用 ubuntu 使用者，而非 root。

3. 執行下列命令，檢查 TPM 裝置是否存在並正常運作：

```
ls -la /dev/tpm*
```

您應該會看到 `/dev/tpm0` 和 `/dev/tpmrm0` 裝置。

步驟 2：安裝和設定 TPM 工具

1. 執行下列命令來安裝所需的套件：

```
sudo apt update
sudo apt install tpm2-openssl tpm2-tools tpm2-abrmd libtss2-tcti-tabrmd0
```

2. 執行下列命令來驗證 TPM 裝置的許可：

```
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
```

步驟 3：設定 OpenSSL TPM2 供應商

1. 編輯 OpenSSL 組態檔案：

```
sudo vi /etc/ssl/openssl.cnf
```

2. 新增下列組態：

```
[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. 視需要調整模組路徑。您可以使用下列方式找到正確的路徑：

```
find /usr -name "tpm2.so"
```

步驟 4：產生持久性 TPM 金鑰

1. 執行下列命令來建立主索引鍵：

```
sudo tpm2_createprimary -C o -c primary.ctx
```

2. 執行下列命令來建立 ECC 金鑰物件：

```
sudo tpm2_create -C primary.ctx -g sha256 -G ecc256 -r device.priv -u device.pub
```

3. 執行下列命令載入金鑰：

```
sudo tpm2_load -C primary.ctx -r device.priv -u device.pub -c device.ctx
```

4. 執行下列命令，使金鑰持久性：

```
sudo tpm2_evictcontrol -C o -c device.ctx 0x81000002
```

這會建立具有控制代碼的持久性金鑰（例如 0x81000002）。

步驟 5：產生憑證簽署請求 (CSR)

在此步驟中，您將使用 TPM2-protected 私有金鑰來產生憑證簽署請求 (CSR)。

1. 使用 TPM 金鑰產生 CSR：

```
openssl req -new -provider tpm2 -key "handle:0x81000002" \  
-out device.csr \  
-subj "/CN=TPMThing"
```

2. 0x81000002 將取代為您選擇的控制代碼值，並將取代 TPMThing 為所需的物件名稱。

步驟 6：從 CSR 建立憑證

1. 在開發電腦上，建立您下載 AWS IoT 物件憑證的資料夾。

```
mkdir greengrass-v2-certs
```

2. 使用 CSR 檔案來建立 AWS IoT 物件的憑證，並將其下載到您的開發電腦。

```
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
```

如果請求成功，回應看起來與下列範例類似：

```
{  
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificateId":  
  "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificatePem": "-----BEGIN CERTIFICATE-----  
MIICiTCCAFICQD6m7oRw0uX0jANBgqhkiG9w  
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ  
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBA5TC0lBTSBDb25zb2x1MRIw  
EAYDVQQDEw1UZXRhbnQ2IjYwMxHzAdBgqhkiG9w0BCQEWEG5vb251QGFTYXpvi5  
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAkGA1UEBh  
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb  
WF6b24xFDASBgNVBA5TC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXRhbnQ2IjYwMx  
HzAdBgqhkiG9w0BCQEWEG5vb251QGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE  
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI  
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ  
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr  
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN  
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJlJ00zbhNYS5f6Guo  
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw  
3rrszlaEXAMPLE=  
-----END CERTIFICATE-----"  
}
```

步驟 7：使用 TPM 支援設定 Greengrass Nucleus Lite

若要在 Greengrass Nucleus Lite 中啟用 TPM 支援，請進行下列變更：

1. 透過將使用者新增至 `tss` 群組以進行 TPM 存取來設定 `ggcore` 使用者許可：

```
sudo usermod -a -G tss ggcore
```

2. 完成下列步驟以更新登入資料目錄：

- 從登入資料目錄移除私有金鑰檔案。
- 由於我們使用持久性 TPM 金鑰，因此不需要複製私有金鑰檔案。
- 將新的 複製到該登入 `device.pem.crt` 資料目錄。

3. `config.yaml` 使用以下 TPM 特定組態進行編輯：

```
system:
  privateKeyPath: "handle:0x81000002" # Use your chosen handle
  certificateFilePath: "" # Replace with the path of device.pem.crt
  ...
```

4. 執行下列命令以重新啟動 Greengrass Nucleus Lite：

```
systemctl restart greengrass-lite.target
```

疑難排解

如果您在設定或操作 TPM2-enabled AWS IoT Greengrass 的裝置期間遇到問題，請嘗試下列疑難排解步驟：

找不到 TPM 裝置（使用 NitroTPM 時）

如果 `/dev/tpm0` 不存在，請執行下列步驟：

1. 確認您使用的是 NitroTPM 支援的執行個體類型。
2. 確保使用 `建立 AMI --tpm-support v2.0`。
3. 確認執行個體已從自訂 AMI 啟動。

拒絕許可錯誤

如果您遇到 TPM 存取錯誤，請執行下列動作：

1. 確認使用者位於 tss 群組中：groups \$USER。
2. 使用以下命令檢查 TPM 裝置的許可：

```
ls -l /dev/tpm*
```

3. 確認您已登出，並在新增至 tss 群組後重新登入。

OpenSSL 提供者問題

如果找不到 TPM 提供者，請執行下列動作：

1. 在中驗證tpm2.so路徑/etc/ssl/openssl.cnf。
2. 執行下列命令來檢查提供者安裝：

```
openssl list -providers
```

3. 確認tpm2-openssl套件已正確安裝。

後續步驟

現在您已成功將 AWS IoT Greengrass 核心裝置與 TPM2 整合，您可以：

- 將元件部署到您的安全 AWS IoT Greengrass 裝置
- 使用 TPM2 整合設定其他 AWS IoT Greengrass 裝置

如需 AWS IoT Greengrass 裝置安全性的詳細資訊，請參閱 [中的安全性 AWS IoT Greengrass](#)。

教學課程：開始使用 SageMaker AI Edge Manager

Important

SageMaker AI Edge Manager 已於 2024 年 4 月 26 日終止。如需繼續將模型部署到邊緣裝置的詳細資訊，請參閱 [SageMaker AI Edge Manager 生命週期結束](#)。

Amazon SageMaker AI Edge Manager 是一種在邊緣裝置上執行的軟體代理程式。SageMaker AI Edge Manager 為邊緣裝置提供模型管理，讓您可以直接在 Greengrass 核心裝置上封裝和使用 Amazon SageMaker AI Neo 編譯的模型。透過使用 SageMaker AI Edge Manager，您也可以從核心裝置取樣模型輸入和輸出資料，並將該資料傳送至 AWS 雲端以進行監控和分析。如需 SageMaker AI Edge Manager 如何在 Greengrass 核心裝置上運作的詳細資訊，請參閱 [在 Greengrass 核心裝置上使用 Amazon SageMaker AI Edge Manager](#)。

本教學課程說明如何開始使用 SageMaker AI Edge Manager 搭配現有核心裝置上 AWS 提供的範例元件。這些範例元件使用 SageMaker AI Edge Manager 元件做為相依性來部署 Edge Manager 代理程式，並使用使用 SageMaker AI Neo 編譯的預先訓練模型執行推論。如需 SageMaker AI Edge Manager 代理程式的詳細資訊，請參閱《Amazon [SageMaker AI 開發人員指南](#)》中的 [SageMaker AI Edge Manager](#)。Amazon SageMaker

若要在現有的 Greengrass 核心裝置上設定和使用 SageMaker AI Edge Manager 代理程式，AWS 會提供範例程式碼，供您用來建立下列範例推論和模型元件。

- 影像分類
 - `com.greengrass.SageMakerEdgeManager.ImageClassification`
 - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- 物件偵測
 - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
 - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

本教學課程說明如何部署範例元件和 SageMaker AI Edge Manager 代理程式。

主題

- [先決條件](#)
- [在 SageMaker AI Edge Manager 中設定 Greengrass 核心裝置](#)
- [建立範例元件](#)
- [執行範例影像分類推論](#)

先決條件

若要完成本教學課程，您必須符合下列先決條件：

- 在 Amazon Linux 2 上執行的 Greengrass 核心裝置、以 Debian 為基礎的 Linux 平台 (x86_64 或 Armv8) 或 Windows (x86_64)。如果您沒有帳戶，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。
- [Python](#) 3.6 或更新版本，包括 pip 安裝在核心裝置上的 Python 版本。
- 安裝在核心裝置上的 OpenGL API GLX 執行期 (libgl1-mesa-glx)。
- 具有管理員許可的 AWS Identity and Access Management (IAM) 使用者。
- 具備網際網路功能的 Windows、Mac 或類似 Unix 的開發電腦，符合下列需求：
 - 已安裝 [Python](#) 3.6 或更新版本。
 - AWS CLI 已安裝並使用 IAM 管理員使用者登入資料進行設定。如需詳細資訊，請參閱 [安裝 AWS CLI](#) 和 [設定 AWS CLI](#)。
- 在與 Greengrass 核心裝置相同的 AWS 帳戶 和 中建立 AWS 區域 的下列 S3 儲存貯體：
 - 存放範例推論和模型元件中包含的成品的 S3 儲存貯體。本教學課程使用 amzn-s3-demo-bucket1 來參考此儲存貯體。
 - 與 SageMaker AI 邊緣裝置機群相關聯的 S3 儲存貯體。SageMaker AI Edge Manager 需要 S3 儲存貯體來建立邊緣裝置機群，並儲存範例資料，避免在裝置上執行推論。本教學課程使用 amzn-s3-demo-bucket2 來參考此儲存貯體。

如需建立 S3 儲存貯體的資訊，請參閱 [Amazon S3 入門](#)。

- 使用下列項目設定的 [Greengrass 裝置角色](#)：
 - 允許 `credentials.iot.amazonaws.com` 和 `sagemaker.amazonaws.com` 擔任角色的信任關係，如下列 IAM 政策範例所示。

```
{
  "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"
    }
  ]
}
```

```
    "Action": "sts:AssumeRole"
  }
]
}
```

- [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM 受管政策。
- [AmazonSageMakerFullAccess](#) IAM 受管政策。
- 包含元件成品之 S3 儲存貯體 s3:GetObject 的動作，如下列 IAM 政策範例所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

在 SageMaker AI Edge Manager 中設定 Greengrass 核心裝置

SageMaker AI Edge Manager 中的邊緣裝置機群是邏輯分組裝置的集合。若要搭配 SageMaker AI Edge Manager 使用 AWS IoT Greengrass，您必須建立邊緣裝置機群，其使用與您部署 SageMaker AI Edge Manager 代理程式的 Greengrass 核心裝置相同的 AWS IoT 角色別名。然後，您必須將核心裝置註冊為該機群的一部分。

主題

- [建立邊緣裝置機群](#)
- [註冊您的 Greengrass 核心裝置](#)

建立邊緣裝置機群

建立邊緣裝置機群（主控台）

1. 在 [Amazon SageMaker AI 主控台](#) 中，選擇 Edge Manager，然後選擇 Edge 裝置機群。
2. 在裝置機群頁面上，選擇建立裝置機群。
3. 在裝置機群屬性下，執行下列動作：
 - 針對裝置機群名稱，輸入裝置機群的名稱。
 - 針對 IAM 角色，輸入您在設定 Greengrass 核心裝置時所指定 AWS IoT 角色別名的 Amazon Resource Name (ARN)。
 - 停用建立 IAM 角色別名切換。
4. 選擇下一步。
5. 在輸出組態下，針對 S3 儲存貯體 URI，輸入您要與裝置機群建立關聯的 S3 儲存貯體 URI。
6. 選擇提交。

註冊您的 Greengrass 核心裝置

將 Greengrass 核心裝置註冊為邊緣裝置（主控台）

1. 在 [Amazon SageMaker AI 主控台](#) 中，選擇 Edge Manager，然後選擇 Edge 裝置。
2. 在裝置頁面上，選擇註冊裝置。
3. 在裝置屬性下，針對裝置機群名稱，輸入您建立的裝置機群名稱，然後選擇下一步。
4. 選擇下一步。
5. 在裝置來源下，針對裝置名稱，輸入 Greengrass 核心裝置的 AWS IoT 物件名稱。
6. 選擇提交。

建立範例元件

為了協助您開始使用 SageMaker AI Edge Manager 元件，會在 GitHub 上 AWS 提供 Python 指令碼，以建立範例推論和模型元件，並 AWS 雲端 為您將其上傳至。在開發電腦上完成下列步驟。

建立範例元件

1. 將 GitHub 上的 [AWS IoT Greengrass 元件範例](#) 儲存庫下載到您的開發電腦。

- 導覽至下載的/machine-learning/sagemaker-edge-manager資料夾。

```
cd download-directory/machine-learning/sagemaker-edge-manager
```

- 執行下列命令，以建立範例元件並將其上傳至 AWS 雲端。

```
python3 create_components.py -r region -b amzn-s3-demo-bucket
```

將*##*取代為您建立 Greengrass 核心裝置的 AWS 區域，並將 amzn-s3-demo-bucket1 取代為 S3 儲存貯體的名稱，以存放元件成品。

Note

根據預設，指令碼會建立影像分類和物件偵測推論的範例元件。若要僅針對特定類型的推論建立元件，請指定 *-i ImageClassification | ObjectDetection* 引數。

用於 SageMaker AI Edge Manager 的範例推論和模型元件現在會在您的 中建立 AWS 帳戶。若要在[AWS IoT Greengrass 主控台](#)中查看範例元件，請選擇元件，然後在我的元件下搜尋下列元件：

- com.greengrass.SageMakerEdgeManager.ImageClassification
- com.greengrass.SageMakerEdgeManager.ImageClassification.Model
- com.greengrass.SageMakerEdgeManager.ObjectDetection
- com.greengrass.SageMakerEdgeManager.ObjectDetection.Model

執行範例影像分類推論

若要使用 AWS 提供的範例元件和 SageMaker AI Edge Manager 代理程式執行映像分類推論，您必須將這些元件部署到核心裝置。部署這些元件會下載 SageMaker AI Neo 編譯的預先訓練 Resnet-50 模型，並在您的裝置上安裝 SageMaker AI Edge Manager 代理程式。SageMaker AI Edge Manager 代理程式會載入模型，並在 gg/sageMakerEdgeManager/image-classification 主題上發佈推論結果。若要檢視這些推論結果，請使用 AWS IoT 主控台中的 AWS IoT MQTT 用戶端來訂閱此主題。

主題

- [訂閱通知主題](#)
- [部署範例元件](#)
- [檢視推論結果](#)

訂閱通知主題

在此步驟中，您會在 AWS IoT 主控台中設定 AWS IoT MQTT 用戶端，以監看範例推論元件發佈的 MQTT 訊息。根據預設，元件會在 `gg/sageMakerEdgeManager/image-classification` 主題上發佈推論結果。在您將元件部署到 Greengrass 核心裝置之前，請訂閱此主題，以查看元件第一次執行時的推論結果。

訂閱預設通知主題

1. 在 [AWS IoT 主控台](#) 導覽功能表中，選擇測試、MQTT 測試用戶端。
2. 在訂閱主題下，於主題名稱方塊中，輸入 **`gg/sageMakerEdgeManager/image-classification`**。
3. 選擇 Subscribe (訂閱)。

部署範例元件

在此步驟中，您會設定下列元件並將其部署到您的核心裝置：

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

若要部署您的元件 (主控台)

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇部署，然後選擇您要修改的目標裝置的部署。
2. 在部署頁面上，選擇修訂，然後選擇修訂部署。
3. 在指定目標頁面上，選擇下一步。
4. 在選取元件頁面上，執行下列動作：
 - a. 在我的元件下，選取下列元件：
 - `com.greengrass.SageMakerEdgeManager.ImageClassification`
 - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
 - b. 在公有元件下，關閉僅顯示選取的元件切換，然後選取 `aws.greengrass.SageMakerEdgeManager` 元件。
 - c. 選擇下一步。

5. 在設定元件頁面上，選取`aws.greengrass.SageMakerEdgeManager`元件並執行下列動作。
 - a. 選擇設定元件。
 - b. 在組態更新下的要合併的組態中，輸入下列組態。

```
{
  "DeviceFleetName": "device-fleet-name",
  "BucketName": "amzn-s3-demo-bucket"
}
```

以您建立的邊緣裝置機群名稱取代 `device-fleet-name`，並以與裝置機群相關聯的 S3 儲存貯體名稱取代 `amzn-s3-demo-bucket`。

- c. 選擇確認，然後選擇下一步。
6. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
7. 在檢閱頁面上，選擇部署

若要部署您的元件 (AWS CLI)

1. 在您的開發電腦上，建立 `deployment.json` 檔案來定義 SageMaker AI Edge Manager 元件的部署組態。該檔案應如以下範例所示。

```
{
  "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": {
      }
    }
  }
}
```

```
    },  
  }  
}
```

- 在 `targetArn` 欄位中，*targetArn* 以下列格式將物件或物群組的 Amazon Resource Name (ARN) 取代為目標部署：
 - 物件：`arn:aws:iot:region:account-id:thing/thingName`
 - 物件群組：`arn:aws:iot:region:account-id:thinggroup/thingGroupName`
- 在 `merge` 欄位中，將 *device-fleet-name* 取代為您建立的邊緣裝置機群名稱。然後，將 *amzn-s3-demo-bucket2* 取代為與裝置機群相關聯的 S3 儲存貯體名稱。
- 以最新的可用版本取代每個元件的元件版本。

2. 執行下列命令以在裝置上部署元件：

```
aws greengrassv2 create-deployment \  
  --cli-input-json file://path/to/deployment.json
```

可能需要幾分鐘才能完成部署。在下一個步驟中，檢查元件記錄檔以確認部署是否已順利完成，並檢視推論結果。

檢視推論結果

部署元件之後，您可以在 Greengrass 核心裝置上的元件日誌中，以及在 AWS IoT 主控台的 AWS IoT MQTT 用戶端中檢視推論結果。若要訂閱元件發佈推論結果的主題，請參閱 [訂閱通知主題](#)。

- AWS IoT MQTT 用戶端 - 若要檢視推論元件在[預設通知主題](#)上發佈的結果，請完成下列步驟：

1. 在[AWS IoT 主控台](#)導覽功能表中，選擇測試、MQTT 測試用戶端。
2. 在訂閱下，選擇 **gg/sageMakerEdgeManager/image-classification**。

- 元件日誌 - 若要在元件日誌中檢視推論結果，請在 Greengrass 核心裝置上執行下列命令。

```
sudo tail -f /greengrass/v2/logs/  
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

如果您在元件日誌或 MQTT 用戶端中看不到推論結果，則部署失敗或無法連線到核心裝置。如果您的核心裝置未連線至網際網路，或沒有執行元件的正確許可，就可能發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass 核心軟體日誌檔案。此檔案包含 Greengrass 核心裝置的部署服務日誌。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

如需詳細資訊，請參閱[故障診斷機器學習推論](#)。

教學課程：使用 TensorFlow Lite 執行範例影像分類推論

本教學課程說明如何使用 [TensorFlow Lite 影像分類](#) 推論元件，在 Greengrass 核心裝置上執行範例影像分類推論。此元件包含下列元件相依性：

- TensorFlow Lite 影像分類模型存放區元件
- TensorFlow Lite 執行時間元件

當您部署此元件時，它會下載預先訓練的 MobileNet v1 模型，並安裝 [TensorFlow Lite](#) 執行時間及其相依性。此元件會在 ml/tflite/image-classification 主題上發佈推論結果。若要檢視這些推論結果，請使用 AWS IoT 主控台內的 AWS IoT MQTT 用戶端來訂閱此主題。

在本教學課程中，您將部署範例推論元件，以在提供的範例映像上執行映像分類 AWS IoT Greengrass。完成本教學課程後，您可以完成 [教學課程：使用 TensorFlow Lite 從相機執行影像分類推論範例](#)，其中說明如何修改範例推論元件，以在 Greengrass 核心裝置上的本機攝影機影像上執行影像分類。

如需 Greengrass 裝置上機器學習的詳細資訊，請參閱 [執行機器學習推論](#)。

主題

- [先決條件](#)
- [步驟 1：訂閱預設通知主題](#)
- [步驟 2：部署 TensorFlow Lite 影像分類元件](#)
- [步驟 3：檢視推論結果](#)
- [後續步驟](#)

先決條件

為了完成本教學，您需要以下項目：

- Linux Greengrass 核心裝置。如果您沒有帳戶，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。核心裝置必須符合下列要求：
 - 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
 - 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

- 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

- 選取界面選項。
- 選取舊版攝影機以啟用舊版攝影機堆疊。
- 重新啟動 Raspberry Pi。

步驟 1：訂閱預設通知主題

在此步驟中，您可以在 AWS IoT 主控台中設定 AWS IoT MQTT 用戶端，以監看 TensorFlow Lite 影像分類元件發佈的 MQTT 訊息。根據預設，元件會在 `ml/tflite/image-classification` 主題上發

佈推論結果。在您將元件部署到 Greengrass 核心裝置之前，請訂閱此主題，以查看元件第一次執行時的推論結果。

訂閱預設通知主題

1. 在[AWS IoT 主控台](#)導覽功能表中，選擇測試、MQTT 測試用戶端。
2. 在訂閱主題下，於主題名稱方塊中，輸入 **ml/tflite/image-classification**。
3. 選擇 Subscribe (訂閱)。

步驟 2：部署 TensorFlow Lite 影像分類元件

在此步驟中，您將 TensorFlow Lite 映像分類元件部署至核心裝置：

部署 TensorFlow Lite 影像分類元件（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
2. 在元面頁面上的公用元件索引標籤上，選擇 `aws.greengrass.TensorFlowLiteImageClassification`。
3. 在 `aws.greengrass.TensorFlowLiteImageClassification` 頁面中，選擇部署。
4. 從新增至部署中，選擇下列其中一項：
 - a. 若要將此元件合併至目標裝置上的現有部署，請選擇新增至現有部署，然後選取要修訂的部署。
 - b. 若要在目標裝置上建立新部署，請選擇建立新部署。如果您的設備上有現有的部署，則選擇此步驟將取代現有部署。
5. 在指定目標頁面上，執行下列作業：
 - a. 在部署資訊下，輸入或修改部署的易記名稱。
 - b. 在部署目標下，選取部署的目標，然後選擇下一步。如果您要修訂既有部署，則無法變更部署目標。
6. 在選取元件頁面的公有元件下，確認已選取 `aws.greengrass.TensorFlowLiteImageClassification` 元件，然後選擇下一步。
7. 在設定元件頁面上，保留預設組態設定，然後選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
9. 在檢閱頁面上，選擇部署

部署 TensorFlow Lite 影像分類元件 (AWS CLI)

1. 建立 `deployment.json` 檔案以定義 TensorFlow Lite 影像分類元件的部署組態。此檔案看起來應該如下所示：

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```

- 在 `targetArn` 欄位中，*targetArn* 以下列格式將物件或物件組的 Amazon Resource Name (ARN) 取代為目標部署：
 - 物件：`arn:aws:iot:region:account-id:thing/thingName`
 - 物件組：`arn:aws:iot:region:account-id:thinggroup/thingGroupName`
- 本教學課程使用元件 2.1.0 版。
在 `aws.greengrass.TensorFlowLiteObjectDetection` 元件物件中，取代 *2.1.0* 以使用不同版本的 TensorFlow Lite 物件偵測元件。

2. 執行下列命令，在裝置上部署 TensorFlow Lite 影像分類元件：

```
aws greengrassv2 create-deployment \
  --cli-input-json file://path/to/deployment.json
```

可能需要幾分鐘才能完成部署。在下一個步驟中，檢查元件記錄檔以確認部署是否已順利完成，並檢視推論結果。

步驟 3：檢視推論結果

部署元件之後，您可以在 Greengrass 核心裝置上的元件日誌中，以及在 AWS IoT 主控台的 AWS IoT MQTT 用戶端中檢視推論結果。若要訂閱元件發佈推論結果的主題，請參閱 [步驟 1：訂閱預設通知主題](#)。

- AWS IoT MQTT 用戶端 - 若要檢視推論元件在 [預設通知主題](#) 上發佈的結果，請完成下列步驟：

1. 在[AWS IoT 主控台](#)導覽功能表中，選擇測試、MQTT 測試用戶端。
2. 在訂閱下，選擇 **ml/tflite/image-classification**。

您應該會看到類似下列範例的訊息。

```
{
  "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"
    }
  ]
}
```

- 元件日誌 - 若要在元件日誌中檢視推論結果，請在 Greengrass 核心裝置上執行下列命令。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

您應該會看到類似下列範例的結果。


```
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}
```

如果您在元件日誌或 MQTT 用戶端中看不到推論結果，則部署會失敗或無法連線到核心裝置。如果您的核心裝置未連線至網際網路，或沒有執行元件的正確許可，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass 核心軟體日誌檔案。此檔案包含 Greengrass 核心裝置部署服務的日誌。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

如需詳細資訊，請參閱[故障診斷機器學習推論](#)。

後續步驟

如果您的 Greengrass 核心裝置具有支援的攝影機界面，您可以完成 [教學課程：使用 TensorFlow Lite 從相機執行影像分類推論範例](#)，這說明如何修改範例推論元件，以對攝影機的影像執行影像分類。

若要進一步探索範例 [TensorFlow Lite 影像分類](#) 推論元件的組態，請嘗試下列動作：

- 修改 InferenceInterval 組態參數，以變更推論程式碼執行的頻率。
- 修改推論元件 ImageDirectory 組態中的 ImageName 和 組態參數，以指定用於推論的自訂映像。

如需自訂公有元件組態或建立自訂機器學習元件的詳細資訊，請參閱 [自訂您的機器學習元件](#)。

教學課程：使用 TensorFlow Lite 從相機執行影像分類推論範例

本教學課程說明如何使用 [TensorFlow Lite 影像分類](#) 推論元件，在 Greengrass 核心裝置上的本機攝影機影像上執行範例影像分類推論。此元件包含下列元件相依性：

- TensorFlow Lite 影像分類模型存放區元件
- TensorFlow Lite 執行時間元件

Note

本教學課程會存取 [Raspberry Pi](#) 或 [NVIDIA Jetson Nano](#) 裝置的攝影機模組，但 AWS IoT Greengrass 支援 Armv7l, Armv8 或 x86_64 平台上的其他裝置。若要為不同的裝置設定攝影機，請參閱裝置的相關文件。

如需 Greengrass 裝置上機器學習的詳細資訊，請參閱 [執行機器學習推論](#)。

主題

- [先決條件](#)
- [步驟 1：在您的裝置上設定攝影機模組](#)
- [步驟 2：驗證預設通知主題的訂閱](#)
- [步驟 3：修改 TensorFlow Lite 影像分類元件組態並進行部署](#)
- [步驟 4：檢視推論結果](#)
- [後續步驟](#)

先決條件

若要完成本教學課程，您必須先完成 [教學課程：使用 TensorFlow Lite 執行範例影像分類推論](#)。

您也需要下列項目：

- 具有相機界面的 Linux Greengrass 核心裝置。本教學課程存取下列支援裝置之一上的攝影機模組：
 - 執行 [Raspberry Pi](#) OS 的 Raspberry Pi（先前稱為 Raspbian）<https://www.raspberrypi.org/downloads/>
 - [NVIDIA Jetson Nano](#)

如需設定 Greengrass 核心裝置的資訊，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。

核心裝置必須符合下列要求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
 3. 選取舊版攝影機以啟用舊版攝影機堆疊。
 4. 重新啟動 Raspberry Pi。
- 對於 Raspberry Pi 或 NVIDIA Jetson Nano 裝置，[Raspberry Pi 相機模組 V2 - 8 百萬像素，1080p](#)。若要了解如何設定相機，請參閱 Raspberry Pi 文件中的[連接相機](#)。

步驟 1：在您的裝置上設定攝影機模組

在此步驟中，您會為裝置安裝並啟用攝影機模組。在裝置上執行下列命令。

Raspberry Pi (Arm v7l)

1. 安裝攝影機模組的picamera界面。執行下列命令來安裝相機模組和本教學課程所需的其他 Python 程式庫。

```
sudo apt-get install -y python3-picamera
```

2. 確認 Picamera 安裝成功。

```
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

如果輸出中未包含錯誤，則表示驗證成功。

Note

如果您的裝置上安裝的 Python 可執行檔是 python3.7，請使用 python3.7，而不是本教學課程中python3的命令。確保您的 pip 安裝對應到正確的 python3.7 或 python3 版本，以避免相依性錯誤。

3. 重新啟動裝置。

```
sudo reboot
```

4. 請開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

5. 請使用箭頭鍵開啟 Interfacing Options (連接選項) 並啟用攝影機界面。如果出現提示，請允許重新啟動裝置。
6. 執行下列命令來測試攝影機設定。

```
raspistill -v -o test.jpg
```

這會在 Raspberry Pi 上開啟預覽視窗、將名為 test.jpg 的圖片儲存至現行目錄，並在 Raspberry Pi 終端機中顯示相機的相關資訊。

7. 執行下列命令來建立符號連結，讓推論元件從執行期元件建立的虛擬環境存取您的攝影機。

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/  
greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

本教學課程的 *MLRootPath* 預設值為 */greengrass/v2/work/variant.TensorFlowLite/greengrass_ml*。當您第一次在 中部署推論元件時，就會建立此位置的greengrass_ml_tflite_venv資料夾[教學課程：使用 TensorFlow Lite 執行範例影像分類推論](#)。

Jetson Nano (Armv8)

1. 執行下列命令來測試攝影機設定。

```
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
```

這會擷取名為 `test.jpg` 的映像，並將其儲存至 `test.jpg` 您目前的目錄。

2. （選用）重新啟動裝置。如果您在上一個步驟中執行 `gst-launch` 命令時遇到問題，重新啟動裝置可能會解決這些問題。

```
sudo reboot
```

Note

對於 Armv8 (AArch64) 裝置，例如 Jetson Nano，您不需要建立符號連結，即可讓推論元件從執行時間元件建立的虛擬環境中存取攝影機。

步驟 2：驗證預設通知主題的訂閱

在 [中教學課程：使用 TensorFlow Lite 執行範例影像分類推論](#)，您已設定 AWS IoT MQTT 用戶端在 AWS IoT 主控台中設定為監看主題上 TensorFlow Lite 影像分類元件發佈的 MQTT 訊息 `ml/tflite/image-classification`。在 AWS IoT 主控台中，驗證此訂閱是否存在。如果沒有，請遵循中的步驟[步驟 1：訂閱預設通知主題](#)訂閱此主題，然後再將元件部署到 Greengrass 核心裝置。

步驟 3：修改 TensorFlow Lite 影像分類元件組態並進行部署

在此步驟中，您會設定 TensorFlow Lite 影像分類元件並將其部署至核心裝置：

設定和部署 TensorFlow Lite 影像分類元件（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
2. 在元面頁面上的公用元件索引標籤上，選擇 `aws.greengrass.TensorFlowLiteImageClassification`。
3. 在 `aws.greengrass.TensorFlowLiteImageClassification` 頁面中，選擇部署。
4. 從新增至部署中，選擇下列其中一項：
 - a. 若要將此元件合併至目標裝置上的現有部署，請選擇新增至現有部署，然後選取要修訂的部署。
 - b. 若要在目標裝置上建立新部署，請選擇建立新部署。如果您的設備上有現有的部署，則選擇此步驟將取代現有部署。
5. 在指定目標頁面上，執行下列作業：
 - a. 在部署資訊下，輸入或修改部署的易記名稱。
 - b. 在部署目標下，選取部署的目標，然後選擇下一步。如果您要修訂既有部署，則無法變更部署目標。
6. 在選取元件頁面的公有元件下，確認已選取 `aws.greengrass.TensorFlowLiteImageClassification` 元件，然後選擇下一步。
7. 在設定元件頁面上，執行下列動作：
 - a. 選取推論元件，然後選擇設定元件。
 - b. 在組態更新下，在要合併的組態方塊中輸入下列組態更新。

```
{
  "InferenceInterval": "60",
  "UseCamera": "true"
}
```

在此組態更新中，元件會存取您裝置上的攝影機模組，並對攝影機拍攝的影像執行推論。推論程式碼每 60 秒執行一次。

- c. 選擇確認，然後選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

9. 在檢閱頁面上，選擇部署

設定和部署 TensorFlow Lite 影像分類元件 (AWS CLI)

1. 建立 `deployment.json` 檔案以定義 TensorFlow Lite 影像分類元件的部署組態。此檔案看起來應該如下所示：

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
    }
  }
}
```

- 在 `targetArn` 欄位中，*targetArn* 以下列格式將物件或物群組的 Amazon Resource Name (ARN) 取代為目標部署：
 - 物件：`arn:aws:iot:region:account-id:thing/thingName`
 - 物件群組：`arn:aws:iot:region:account-id:thinggroup/thingGroupName`
- 本教學課程使用元件 2.1.0 版。
在 `aws.greengrass.TensorFlowLiteImageClassification` 元件物件中，取代 *2.1.0* 以使用不同版本的 TensorFlow Lite 影像分類元件。

在此組態更新中，元件會存取您裝置上的攝影機模組，並對攝影機拍攝的影像執行推論。推論程式碼每 60 秒執行一次。取代下列值

2. 執行下列命令，在裝置上部署 TensorFlow Lite 影像分類元件：

```
aws greengrassv2 create-deployment \  
  --cli-input-json file://path/to/deployment.json
```

可能需要幾分鐘才能完成部署。在下一個步驟中，檢查元件記錄檔以確認部署是否已順利完成，並檢視推論結果。

步驟 4：檢視推論結果

部署元件之後，您可以在 Greengrass 核心裝置上的元件日誌中，以及在 AWS IoT 主控台的 AWS IoT MQTT 用戶端中檢視推論結果。若要訂閱元件發佈推論結果的主題，請參閱 [步驟 2：驗證預設通知主題的訂閱](#)。

- AWS IoT MQTT 用戶端 - 若要檢視推論元件在[預設通知主題](#)上發佈的結果，請完成下列步驟：
 1. 在[AWS IoT 主控台](#)導覽功能表中，選擇測試、MQTT 測試用戶端。
 2. 在訂閱下，選擇 **ml/tflite/image-classification**。
- 元件日誌 - 若要在元件日誌中檢視推論結果，請在 Greengrass 核心裝置上執行下列命令。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

如果您在元件日誌或 MQTT 用戶端中看不到推論結果，則部署失敗或無法連線到核心裝置。如果您的核心裝置未連線至網際網路，或沒有執行元件所需的許可，就會發生這種情況。在核心裝置上執行下列命令，以檢視 AWS IoT Greengrass 核心軟體日誌檔案。此檔案包含 Greengrass 核心裝置的部署服務日誌。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

如需詳細資訊，請參閱[故障診斷機器學習推論](#)。

後續步驟

本教學課程說明如何使用 TensorFlow Lite 影像分類元件，搭配自訂組態選項，在相機拍攝的影像上執行範例影像分類。

如需自訂公有元件組態或建立自訂機器學習元件的詳細資訊，請參閱[自訂您的機器學習元件](#)。

元件

AWS IoT Greengrass 元件是您部署到 Greengrass 核心裝置的軟體模組。元件可以代表應用程式、執行期安裝程式、程式庫，或您會在裝置上執行的任何程式碼。您可以定義依賴其他元件的元件。例如，您可以定義安裝 Python 的元件，然後將該元件定義為執行 Python 應用程式之元件的相依性。當您將元件部署到裝置機群時，Greengrass 只會部署裝置所需的軟體模組。

主題

- [AWS提供的元件](#)
- [發行者支援的元件](#)
- [社群元件](#)
- [AWS IoT Greengrass 開發工具](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)

AWS提供的元件

AWS IoT Greengrass 提供和維護預先建置的元件，您可以將這些元件部署到您的裝置。這些元件包括功能（例如串流管理員）、AWS IoT Greengrass V1 連接器（例如 CloudWatch 指標）和本機開發工具（例如 AWS IoT Greengrass CLI）。您可以將[這些元件部署](#)到裝置以進行其獨立功能，也可以將其用作[自訂 Greengrass 元件](#)中的相依項。

Note

數個 AWS 提供的元件取決於 Greengrass 核的特定次要版本。由於此相依性，當您將 Greengrass 核更新為新的次要版本時，您需要更新這些元件。如需每個元件所依賴之核的特定版本的相關資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

當元件同時具有通用和 Lambda 的元件類型時，元件的目前版本是通用類型，而元件的先前版本是 Lambda 類型。

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
Greengrass 核	Core AWS IoT Greengrass 軟體的核。使用此元件來設定和更新核心裝置上的軟體。	Huleus	Linux、Windows	是	否
Greengrass nucleus lite	針對低成本、邊緣裝置和大量應用程式最佳化之資源受限裝置的輕量型核心	NucleusLite	Linux	是	否
用戶端裝置身分驗證	讓稱為用戶端裝置的本機 IoT 裝置連線至核心裝置。	外掛程式	Linux、Windows	是	否
CloudWatch 指標	將自訂指標發佈至 Amazon CloudWatch。	一般、Lambda	Linux、Windows	是	是
AWS IoT Device Defender	通知管理員 Greengrass 核心裝置狀態的變更，	一般、Lambda	Linux、Windows	是	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
	以識別異常行為。				
磁碟多工緩衝處理	針對從 Greengrass 核心裝置到的多工緩衝處理訊息，啟用持久性儲存選項 AWS IoT Core。此元件會將這些傳出訊息存放在磁碟上。	外掛程式	Linux、Windows	是	否
Docker 應用程式管理員	AWS IoT Greengrass 可讓您從 Docker Hub 和 Amazon Elastic Container Registry (Amazon ECR) 下載 Docker 映像。	一般	Linux、Windows	否	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
Kinesis Video Streams 的邊緣連接器	從本機攝影機讀取影片摘要、將串流發佈至 Kinesis Video Streams，並使用 AWS IoT TwinMaker 在 Grafana 儀表板中顯示串流。	一般	Linux	否	否
Greengrass CLI	提供命令列界面，您可以用來建立本機部署，並與 Greengrass 核心裝置及其元件互動。	外掛程式	Linux、Windows	是	否
IP 偵測器	向報告 MQTT 代理程式連線資訊 AWS IoT Greengrass，以使用戶端裝置可以探索如何連線。	外掛程式	Linux、Windows	是	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
Firehose	透過 Amazon Data Firehose 交付串流將資料發佈至中的目的地 AWS 雲端。	Lambda	Linux	否	否
Lambda 啟動器	處理 Lambda 函數的程序和環境組態。	一般	Linux	否	否
Lambda 管理員	處理 Lambda 函數的處理程序間通訊和擴展。	外掛程式	Linux	否	否
Lambda 執行期	提供每個 Lambda 執行時間的成品。	一般	Linux	否	否
傳統訂閱路由器	管理在 AWS IoT Greengrass V1 上執行的 Lambda 函數訂閱。	一般	Linux	否	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
本機偵錯主控台	提供本機主控台，可讓您用來偵錯和管理 Greengrass 核心裝置及其元件。	外掛程式	Linux、Windows	是	否
日誌管理員	在 Greengrass 核心裝置上收集和上傳日誌。	外掛程式	Linux、Windows	是	否
機器學習元件	提供機器學習模型和範例推論程式碼，您可以用來在 Greengrass 核心裝置上執行機器學習推論。	請參閱 機器學習元件 。			否
Modbus-RTU 通訊協定轉接器	從本機 Modbus RTU 裝置輪詢資訊。	Lambda	Linux	否	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
Nucleus 遙測發射器	將從核收集的系統運作狀態遙測資料發佈至本機主題或 AWS IoT Core MQTT 主題。	外掛程式	Linux、Windows	是	否
MQTT 橋接器	在用戶端裝置、本機 AWS IoT Greengrass 發佈/訂閱和之間轉送 MQTT 訊息 AWS IoT Core。	外掛程式	Linux、Windows	是	否
MQTT 3.1.1 代理程式 (Moquette)	執行 MQTT 3.1.1 代理程式，處理用戶端裝置和核心裝置之間的訊息。	外掛程式	Linux、Windows	是	否
MQTT 5 代理程式 (EMQX)	執行 MQTT 5 代理程式，處理用戶端裝置和核心裝置之間的訊息。	一般	Linux、Windows	否	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
PKCS#11 供應商	讓 Greengrass 元件存取您安全地存放在硬體安全模組 (HSM) 中的私有金鑰和憑證。	外掛程式	Linux	是	否
秘密管理員	從秘密部署 AWS Secrets Manager 秘密，讓您可以在 Greengrass 核心裝置上的自訂元件中安全地使用登入資料，例如密碼。	外掛程式	Linux、Windows	是	否
安全通道	啟用 AWS IoT 安全通道連線，您可以使用這些連線與受限防火牆後方的 Greengrass 核心裝置建立競價通訊。	一般	Linux	否	是

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
Shadow 管理員	啟用與核心裝置上陰影的互動。它管理影子文件儲存，以及本機影子狀態與 AWS IoT Device Shadow 服務的同步。	外掛程式	Linux、Windows	是	否
Amazon SNS	發佈訊息至 Amazon SNS 主題。	Lambda	Linux	否	否
串流管理員	將大量資料從本機來源串流至 AWS 雲端。	一般	Linux、Windows	否	是
系統日誌轉寄站	將 systemd-journald 日誌上傳至 AWS 雲端。	一般	Linux	是	是
Systems Manager 代理程式	使用 管理核心裝置 AWS Systems Manager，可讓您修補裝置、執行命令等。	一般	Linux	是	否

元件	說明	元件類型	支援的作業系統	開放原始碼	與 NVMe lite 相容
字符交換服務	提供您可以用來與 AWS 服務互動的 AWS 登入資料。	一般	Linux、Windows	否	否
IoT SiteWise OPC UA 收集器	從 OPC-UA 伺服器收集資料。	一般	Linux、Windows	否	否
IoT SiteWise OPC UA 資料來源模擬器	執行產生範例資料的本機 OPC-UA 伺服器。	一般	Linux、Windows	否	否
IoT SiteWise 發佈者	將資料發佈至 AWS 雲端。	一般	Linux、Windows	否	否
IoT SiteWise 處理器	處理 Greengrass 核心裝置上的資料。	一般	Linux、Windows	否	否

Greengrass 核

Greengrass 核元件 (aws.greengrass.Nucleus) 是強制性元件，也是在裝置上執行 AWS IoT Greengrass Core 軟體的最低需求。您可以設定此元件以從遠端自訂和更新 AWS IoT Greengrass Core 軟體。部署此元件以在核心裝置上設定代理、裝置角色和 AWS IoT 物件組態等設定。

Note

自 Greengrass 2.14.0 版起，限制邊緣裝置可使用記憶體使用量最佳化版本的 nucleus 裝置執行期。如需其組態和使用方式的詳細資訊，請參閱 [Greengrass nucleus lite](#)。

Important

當核心元件的版本變更，或當您變更特定組態參數時，包含您裝置上的核心 AWS IoT Greengrass 軟體和所有其他元件會開始套用變更。

當您部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或更新以這些裝置為目標的部署，則可能會自動將新修補程式版本的 AWS 公有元件部署到您的核心裝置。有些自動更新，例如 核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您[在建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

主題

- [版本](#)
- [裝置要求](#)
- [支援平台](#)
- [作業系統](#)
- [需求](#)
- [相依性](#)
- [下載並安裝](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 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 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

裝置要求

Note

您可以使用 AWS IoT Device Tester 的 AWS IoT Greengrass 來驗證您的裝置是否可以執行 AWS IoT Greengrass 核心軟體並與 通訊 AWS 雲端。如需詳細資訊，請參閱[使用 AWS IoT Device Tester for AWS IoT Greengrass V2](#)。

Linux

- 使用 [AWS 區域](#) 支援的 AWS IoT Greengrass V2。如需支援區域的清單，請參閱《AWS 一般參考》中的 [AWS IoT Greengrass V2 端點和配額](#)。
- Core AWS IoT Greengrass 軟體的可用磁碟空間至少為 256 MB。此要求不包含部署到核心裝置的元件。
- 配置給 AWS IoT Greengrass Core 軟體的最小 96 MB RAM。此要求不包含在核心裝置上執行的元件。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。
- Java 執行期環境 (JRE) 第 8 版或更新版本。Java 必須在裝置上的 [PATH](#) 環境變數上可用。若要使用 Java 開發自訂元件，您必須安裝 Java 開發套件 (JDK)。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
- [GNU C 程式庫](#) (glibc) 2.25 版或更新版本。

- 您必須以根使用者身分執行 AWS IoT Greengrass Core 軟體。例如 `sudo`，使用。
- 執行 AWS IoT Greengrass Core 軟體的根使用者，例如 `root`，必須具有 `sudo` 許可，才能使用任何使用者和任何群組執行。`/etc/sudoers` 檔案必須授予此使用者執行 `sudo` 做為其他群組的許可。中使用者的許可 `/etc/sudoers` 應如下所示。

```
root    ALL=(ALL:ALL) ALL
```

- 核心裝置必須能夠對一組端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。
- `/tmp` 目錄必須掛載有 `exec` 許可。
- 下列所有 shell 命令：
 - `ps -ax -o pid,ppid`
 - `sudo`
 - `sh`
 - `kill`
 - `cp`
 - `chmod`
 - `rm`
 - `ln`
 - `echo`
 - `exit`
 - `id`
 - `uname`
 - `grep`
- 您的裝置也可能需要下列選用的 shell 命令：
 - (選用) `systemctl`。此命令用於將 AWS IoT Greengrass Core 軟體設定為系統服務。
 - (選用) `useradd`、`groupadd` 和 `usermod`。這些命令用於設定 `ggc_user` 系統使用者和 `ggc_group` 系統群組。
 - (選用) `mkfifo`。此命令用於執行 Lambda 函數做為元件。
- 若要設定元件程序的系統資源限制，您的裝置必須執行 Linux 核心版本 2.6.24 或更新版本。
- 若要執行 Lambda 函數，您的裝置必須符合其他要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。

Windows

- 使用 [AWS 區域](#) 支援的 AWS IoT Greengrass V2。如需支援區域的清單，請參閱《AWS 一般參考》中的 [AWS IoT Greengrass V2 端點和配額](#)。
- Core AWS IoT Greengrass 軟體的可用磁碟空間至少為 256 MB。此要求不包含部署到核心裝置的元件。
- 配置給 AWS IoT Greengrass Core 軟體的最小 160 MB RAM。此要求不包含在核心裝置上執行的元件。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。
- Java 執行期環境 (JRE) 第 8 版或更新版本。Java 必須在裝置上的 [PATH](#) 系統變數上可用。若需要使用 Java 開發自訂元件，您必須安裝 Java 開發套件 (JDK)。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。

Note

若要使用 [Greengrass 核](#) 的 2.5.0 版，您必須使用 64 位元版本的 Java 執行期環境 (JRE)。Greengrass 核版本 2.5.1 支援 32 位元和 64 位元 JREs。

- 安裝 AWS IoT Greengrass Core 軟體的使用者必須是管理員。
- 您必須將 AWS IoT Greengrass Core 軟體安裝為系統服務。在安裝軟體 `--setup-system-service true` 時指定。
- 每個執行元件程序的使用者都必須存在於 LocalSystem 帳戶中，而且使用者名稱和密碼必須位於 LocalSystem 帳戶的 Credential Manager 執行個體中。您可以在遵循[安裝 AWS IoT Greengrass Core 軟體](#)的指示時設定此使用者。
- 核心裝置必須能夠對一組端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

支援平台

AWS IoT Greengrass 正式支援執行下列平台的裝置。此清單中未包含平台的裝置可能可以運作，但只能在這些指定的平台上 AWS IoT Greengrass 進行測試。

Linux

架構：

- Armv7l
- Armv8 (AArch64)

- x86_64

Windows

架構：

- x86_64

版本：

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

Note

Windows 裝置目前不支援某些 AWS IoT Greengrass 功能。如需詳細資訊，請參閱[Greengrass 功能相容性及功能考量](#)。

功能考量

Windows 裝置目前不支援某些 AWS IoT Greengrass 功能。檢閱功能差異，以確認 Windows 裝置是否符合您的需求。如需詳細資訊，請參閱[Greengrass 功能相容性](#)。

若要建置自訂 Linux 作業系統，您可以在[meta-aws 專案](#)中使用適用於 AWS IoT Greengrass 的 BitBake 配方。meta-aws 專案提供配方，可讓您在以 [OpenEmbedded](#) 和 Yocto Project 組建架構建置的[嵌入式 Linux](#) 系統中建置 AWS 邊緣軟體功能。[Yocto 專案](#)是一種開放原始碼協同合作專案，可協助您為嵌入式應用程式建置自訂 Linux 型系統，而不論硬體架構為何。的 AWS IoT Greengrass BitBake 配方會在您的裝置上安裝、設定和自動執行 AWS IoT Greengrass Core 軟體。

Linux 平台也可以 AWS IoT Greengrass 在 Docker 容器中執行。如需詳細資訊，請參閱[在 Docker 容器中執行 AWS IoT Greengrass 核心軟體](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

如需詳細資訊，請參閱[支援平台](#)。

需求

裝置必須符合特定需求，才能安裝和執行 Greengrass 核和 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱[裝置要求](#)。

Greengrass 核元件支援在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。

- Greengrass 核元件必須具有與 AWS IoT data、AWS IoT Credentials 和 Amazon S3 的連線能力。

相依性

Greengrass 核不包含任何元件相依性。不過，數個 AWS 提供的元件包含核心做為相依性。如需詳細資訊，請參閱[AWS 提供的元件](#)。

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

下載並安裝

您可以下載安裝程式，在裝置上設定 Greengrass 核元件。此安裝程式會將您的裝置設定為 Greengrass 核心裝置。您可以執行兩種類型的安裝：為您建立必要 AWS 資源的快速安裝，或是自行建立 AWS 資源的手動安裝。如需詳細資訊，請參閱[安裝 AWS IoT Greengrass Core 軟體](#)。

您也可以遵循教學課程來安裝 Greengrass 核，並探索 Greengrass 元件開發。如需詳細資訊，請參閱[教學課程：入門 AWS IoT Greengrass V2](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。有些參數需要 AWS IoT Greengrass Core 軟體重新啟動才能生效。如需為何及如何設定此元件的詳細資訊，請參閱[設定 AWS IoT Greengrass 核心軟體](#)。

iotRoleAlias

指向字符交換 IAM AWS IoT 角色的角色別名。AWS IoT 登入資料提供者會擔任此角色，以允許 Greengrass 核心裝置與 AWS 服務互動。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

當您使用 `--provision true` 選項執行 AWS IoT Greengrass Core 軟體時，軟體會佈建角色別名，並在核心元件中設定其值。

`interpolateComponentConfiguration`

(選用) 您可以啟用 Greengrass 核，在[元件組態中插入元件配方變數](#)，並[合併組態更新](#)。我們建議您將此選項設定為 `true` 以便核心裝置可以執行在其組態中使用配方變數的 Greengrass 元件。

此功能適用於此元件的 v2.6.0 和更新版本。

預設：`false`

`networkProxy`

(選用) 用於所有連線的網路代理。如需詳細資訊，請參閱[連線至連接埠 443 或透過網路代理](#)。

Important

當您部署變更至此組態參數時，AWS IoT Greengrass 核心軟體會重新啟動，讓變更生效。

此物件包含下列資訊：

`noProxyAddresses`

(選用) 從代理中豁免的 IP 地址或主機名稱的逗號分隔清單。

`proxy`

要連線的代理。此物件包含下列資訊：

`url`

代理伺服器的 URL，格式為 `scheme://userinfo@host:port`。

- `scheme` – 結構描述，必須是 `http` 或 `https`。

Important

Greengrass 核心裝置必須執行 [Greengrass 核 v2.5.0](#) 或更新版本，才能使用 HTTPS 代理。

如果您設定 HTTPS 代理，則必須將代理伺服器 CA 憑證新增至核心裝置的 Amazon 根 CA 憑證。如需詳細資訊，請參閱[讓核心裝置信任 HTTPS 代理](#)。

- `userinfo` – (選用) 使用者名稱和密碼資訊。如果您在 `url` 中指定此資訊，Greengrass 核心裝置會忽略 `username` 和 `password` 欄位。
- `host` – 代理伺服器的主機名稱或 IP 地址。
- `port` – (選用) 連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：
 - `http` – 80
 - `https` – 443

`username`

(選用) 驗證代理伺服器的使用者名稱。

`password`

(選用) 驗證代理伺服器的密碼。

`mqtt`

(選用) Greengrass 核心裝置的 MQTT 組態。如需詳細資訊，請參閱[連線至連接埠 443 或透過網路代理](#)。

 Important

當您部署變更至此組態參數時，AWS IoT Greengrass 核心軟體會重新啟動，讓變更生效。

此物件包含下列資訊：

`port`

(選用) 用於 MQTT 連線的連接埠。

預設：8883

`keepAliveTimeoutMs`

(選用) 用戶端為保持 MQTT 連線持續傳送的每個 PING 訊息之間的時間，以毫秒為單位。此值必須大於 `pingTimeoutMs`。

預設：60000(60 秒)

pingTimeoutMs

(選用) 用戶端等待從伺服器接收 PINGACK 訊息的時間，以毫秒為單位。如果等待超過逾時，核心裝置會關閉並重新開啟 MQTT 連線。此值必須小於 keepAliveTimeoutMs。

預設：30000(30 秒)

operationTimeoutMs

(選用) 用戶端等待 MQTT 操作 (例如 CONNECT 或 PUBLISH) 完成的時間，以毫秒為單位。此選項不適用於 MQTT PING 或保持即時訊息。

預設：30000(30 秒)

maxInFlightPublishes

(選用) 可同時傳輸的未確認 MQTT QoS 1 訊息數量上限。

此功能適用於此元件的 v2.1.0 和更新版本。

預設：5

有效範圍：最大值為 100

maxMessageSizeInBytes

(選用) MQTT 訊息的大小上限。如果訊息超過此大小，Greengrass 核會拒絕發生錯誤的訊息。

此功能適用於此元件的 v2.1.0 和更新版本。

預設：131072 (128 KB)

有效範圍：最大值 2621440(2.5 MB)

maxPublishRetry

(選用) 重試無法發佈訊息的次數上限。您可以指定 -1 重試不限時間。

此功能適用於此元件的 v2.1.0 和更新版本。

預設：100

spooler

(選用) Greengrass 核心裝置的 MQTT 多工緩衝處理常式組態。此物件包含下列資訊：

storageType

儲存訊息的儲存類型。如果 storageType 設定為 Disk，pluginName 則可以設定。您可指定為 Memory 或 Disk。

此功能適用於 [Greengrass 核元件](#) 的 v2.11.0 和更新版本。

Important

如果 MQTT 多工緩衝處理器 storageType 設定為 Disk 且您想要將 Greengrass 核從 2.11.x 版降級為舊版，您必須將組態變更回 Memory。Greengrass 核版本 2.10.x 及更早版本 storageType 支援的唯一組態是 Memory。Memory 未遵循此指引可能會導致多工緩衝處理中斷。這會導致您的 Greengrass 核心裝置無法傳送 MQTT 訊息到 AWS 雲端。

預設：Memory

pluginName

(選用) 外掛程式元件名稱。只有在 storageType 設定為 Disk 時，才會使用此元件 Disk。此選項預設為 aws.greengrass.DiskSpooler，並將使用 Greengrass 提供的 [磁碟多工緩衝處理](#)。

此功能適用於 [Greengrass 核元件](#) 的 v2.11.0 和更新版本。

預設："aws.greengrass.DiskSpooler"

maxSizeInBytes

(選用) 核心裝置在記憶體中存放未處理 MQTT 訊息的快取大小上限。如果快取已滿，則會拒絕新訊息。

預設：2621440 (2.5 MB)

keepQos0WhenOffline

(選用) 您可以多工緩衝處理核心裝置離線時接收的 MQTT QoS 0 訊息。如果您將此選項設定為 true，則核心裝置會多工緩衝處理離線時無法傳送的 QoS 0 訊息。如果您將此選項設定為 false，核心裝置會捨棄這些訊息。除非多工緩衝處理已滿，否則核心裝置一律會多工緩衝處理 QoS 1 訊息。

預設：false

version

(選用) MQTT 的版本。您可指定為 mqtt3 或 mqtt5。

此功能適用於 [Greengrass 核元件](#) 的 v2.10.0 和更新版本。

預設：mqtt5

receiveMaximum

(選用) 代理程式可以傳送的未確認 QoS1 封包數目上限。

此功能適用於 [Greengrass 核元件](#) 的 v2.10.0 和更新版本。

預設：100

sessionExpirySeconds

(選用) 您可以請求工作階段從 IoT Core 持續的時間，以秒為單位。預設值為支援的最長時間 AWS IoT Core。

此功能適用於 [Greengrass 核元件](#) 的 v2.10.0 和更新版本。

預設：604800 (7 days)

minimumReconnectDelaySeconds

(選用) 重新連線行為的選項。MQTT 重新連線的最短時間，以秒為單位。

此功能適用於 [Greengrass 核元件](#) 的 v2.10.0 和更新版本。

預設：1

maximumReconnectDelaySeconds

(選用) 重新連線行為的選項。MQTT 重新連線的秒數上限。

此功能適用於 [Greengrass 核元件](#) 的 v2.10.0 和更新版本。

預設：120

minimumConnectedTimeBeforeRetryResetSeconds

(選用) 重新連線行為的選項。在重試延遲重設回最小值之前，連線必須處於作用中狀態的秒數。

此功能適用於 [Greengrass 核元件](#) 的 v2.10.0 和更新版本。

預設：30

jvmOptions

(選用) 用來執行 AWS IoT Greengrass Core 軟體的 JVM 選項。如需執行 AWS IoT Greengrass Core 軟體的建議 JVM 選項相關資訊，請參閱 [使用 JVM 選項控制記憶體配置](#)。

Important

當您部署變更至此組態參數時，AWS IoT Greengrass 核心軟體會重新啟動，讓變更生效。

iotDataEndpoint

您 AWS IoT 的資料端點 AWS 帳戶。

當您使用 `--provision true` 選項執行 AWS IoT Greengrass 核心軟體時，軟體會從取得您的資料和登入資料端點 AWS IoT，並在核心元件中設定它們。

iotCredEndpoint

您的 AWS IoT 登入資料端點 AWS 帳戶。

當您使用 `--provision true` 選項執行 AWS IoT Greengrass 核心軟體時，軟體會從取得您的資料和登入資料端點 AWS IoT，並在核心元件中設定它們。

greengrassDataPlaneEndpoint

此功能適用於此元件的 v2.7.0 和更新版本。

如需詳細資訊，請參閱 [使用私有 CA 簽署的裝置憑證](#)。

greengrassDataPlanePort

此功能適用於此元件的 v2.0.4 和更新版本。

(選用) 用於資料平面連線的连接埠。如需詳細資訊，請參閱 [連線至连接埠 443 或透過網路代理](#)。

Important

您必須指定裝置可以進行傳出連線的连接埠。如果您指定封鎖的连接埠，裝置將無法連線至 AWS IoT Greengrass 以接收部署。

您可以從以下選項中選擇：

- 443
- 8443

預設：8443

`awsRegion`

AWS 區域 要使用的。

`runWithDefault`

用來執行元件的系統使用者。

Important

當您部署變更至此組態參數時，AWS IoT Greengrass 核心軟體會重新啟動，讓變更生效。

此物件包含下列資訊：

`posixUser`

系統使用者的名稱或 ID，也可以選擇核心裝置用來執行一般和 Lambda 元件的系統群組。以下列格式指定使用者和群組，並以冒號 (:) 分隔：`user:group`。群組為選用項目。如果您未指定群組，則 AWS IoT Greengrass Core 軟體會使用使用者的主要群組。例如，您可以指定 `ggc_user` 或 `ggc_user:ggc_group`。如需詳細資訊，請參閱[設定執行元件的使用者](#)。

當您使用 `--component-default-user ggc_user:ggc_group` 選項執行 AWS IoT Greengrass Core 軟體安裝程式時，軟體會在 `nucleus` 元件中設定此參數。

`windowsUser`

此功能適用於此元件的 v2.5.0 和更新版本。

用於在 Windows 核心裝置上執行此元件的 Windows 使用者名稱。使用者必須存在於每個 Windows 核心裝置上，且其名稱和密碼必須存放在 LocalSystem 帳戶的 Credentials Manager 執行個體中。如需詳細資訊，請參閱[設定執行元件的使用者](#)。

當您使用 `--component-default-user ggc_user` 選項執行 AWS IoT Greengrass Core 軟體安裝程式時，軟體會在 `nucleus` 元件中設定此參數。

systemResourceLimits

此功能在 v2.4.0 和此元件的更新版本中提供。AWS IoT Greengrass 目前不支援 Windows 核心裝置上的此功能。

根據預設，要套用至一般和非容器 Lambda 元件程序的系統資源限制。您可以在建立部署時覆寫個別元件的系統資源限制。如需詳細資訊，請參閱[設定元件的系統資源限制](#)。

此物件包含下列資訊：

cpus

每個元件程序可在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定為 2，將每個元件的程序限制為每個 CPU 核心的 50% 用量。在具有 1 個 CPU 核心的裝置上，您可以將此值設定為 0.25，將每個元件的程序限制為 CPU 的 25% 用量。如果您將此值設定為大於 CPU 核心數量的數字，則 AWS IoT Greengrass Core 軟體不會限制元件的 CPU 用量。

memory

每個元件程序可在核心裝置上使用的 RAM 數量上限（以 KB 為單位）。

s3EndpointType

（選用）S3 端點類型。此參數僅適用於美國東部（維吉尼亞北部）(us-east-1) 區域。從任何其他區域設定此參數將被忽略。您可以從以下選項中選擇：

- REGIONAL – S3 用戶端和預先簽章的 URL 使用區域端點。
- GLOBAL – S3 用戶端和預先簽章的 URL 使用舊版端點。
- DUALSTACK – S3 預先簽章的 URL 使用雙堆疊端點。

預設：GLOBAL

fipsMode

（選用）讓 Greengrass 使用 FIPS 端點。如需如何啟用 FIPS 端點的詳細資訊，請參閱[FIPS 端點](#)。

您可以從以下選項中選擇：

- true 當設定為 true 時，端點將使用 FIPS 端點。
- false 當 false 時，端點不會使用 FIPS 端點。

預設：false

logging

(選用) 核心裝置的記錄組態。如需如何設定和使用 Greengrass 日誌的詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

此物件包含下列資訊：

level

(選用) 要輸出的日誌訊息最低層級。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

format

(選用) 日誌的資料格式。您可以從以下選項中選擇：

- TEXT – 如果您想要以文字形式檢視日誌，請選擇此選項。
- JSON – 如果您想要使用 [Greengrass CLI 日誌命令檢視日誌](#) 或以程式設計方式與日誌互動，請選擇此選項。

預設：TEXT

outputType

(選用) 日誌的輸出類型。您可以從以下選項中選擇：

- FILE – Core AWS IoT Greengrass 軟體會將日誌輸出至您在 `outputDirectory` 中指定的目錄中的檔案。
- CONSOLE – Core AWS IoT Greengrass 軟體會將日誌列印到 `stdout`。選擇此選項可在核心裝置列印日誌時檢視日誌。

預設：FILE

fileSizeKB

(選用) 每個日誌檔案的大小上限 (以 KB 為單位)。日誌檔案超過此檔案大小上限後，AWS IoT Greengrass 核心軟體會建立新的日誌檔案。

此參數僅適用於您FILE為 指定 時outputType。

預設：1024

totalLogsSizeKB

(選用) 每個元件的日誌檔案總大小上限 (以 KB 為單位)，包括 Greengrass 核。Greengrass 核的日誌檔案也包含[來自外掛程式元件](#)的日誌。在元件的日誌檔案總大小超過此大小上限之後，AWS IoT Greengrass Core 軟體會刪除該元件最舊的日誌檔案。

此參數等同於[日誌管理員元件的磁碟空間限制](#)參數 (diskSpaceLimit)，您可以為 Greengrass 核 (系統) 和每個元件指定該參數。Core AWS IoT Greengrass 軟體使用兩個值的最小值作為 Greengrass 核和每個元件的總日誌大小上限。

此參數僅適用於您FILE為 指定 時outputType。

預設：10240

outputDirectory

(選用) 日誌檔案的輸出目錄。

此參數僅適用於您FILE為 指定 時outputType。

預設：*/greengrass/v2/logs*，其中 */greengrass/v2* AWS IoT Greengrass 是根資料夾。

fleetstatus

此參數適用於此元件的 v2.1.0 和更新版本。

(選用) 核心裝置的機群狀態組態。

此物件包含下列資訊：

periodicStatusPublishIntervalSeconds

(選用) 核心裝置將裝置狀態發佈到 之間的時間量 (以秒為單位) AWS 雲端。

下限：86400(24 小時)

預設：86400(24 小時)

telemetry

(選用) 核心裝置的系統運作狀態遙測組態。如需遙測指標以及如何對遙測資料採取行動的詳細資訊，請參閱 [從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)。

此物件包含下列資訊：

enabled

(選用) 您可以啟用或停用遙測。

預設：true

periodicAggregateMetricsIntervalSeconds

(選用) 核心裝置彙總指標的間隔 (以秒為單位)。

如果您將此值設定為低於支援的最小值，則 nucleus 會改用預設值。

下限：3600

預設：3600

periodicPublishMetricsIntervalSeconds

(選用) 核心裝置將遙測指標發佈到 之間的時間量 (以秒為單位) AWS 雲端。

如果您將此值設定為低於支援的最小值，則 nucleus 會改用預設值。

下限：86400

預設：86400

deploymentPollingFrequencySeconds

(選用) 輪詢部署通知的期間，以秒為單位。

預設：15

componentStoreMaxSizeBytes

(選用) 元件存放區磁碟上的大小上限，其中包含元件配方和成品。

預設：10000000000 (10 GB)

platformOverride

(選用) 識別核心裝置平台的屬性字典。使用此選項來定義元件配方可用來識別元件正確生命週期和成品的自訂平台屬性。例如，您可以定義硬體功能屬性，只部署要執行之元件的最低成品集。如需詳細資訊，請參閱元件配方中的[資訊清單平台參數](#)。

您也可以使用此參數來覆寫核心裝置的 `os` 和 `architecture` 平台屬性。

httpClient

此參數適用於此元件的 v2.5.0 和更新版本。

(選用) 核心裝置的 HTTP 用戶端組態。這些組態選項適用於此元件提出的所有 HTTP 請求。如果核心裝置在較慢的網路上執行，您可以增加這些逾時持續時間，以防止 HTTP 請求逾時。

此物件包含下列資訊：

connectionTimeoutMs

(選用) 在連線請求逾時之前等待連線開啟的時間量 (以毫秒為單位)。

預設：2000(2 秒)

socketTimeoutMs

(選用) 在連線逾時之前，等待資料透過開放連線傳輸的時間量 (以毫秒為單位)。

預設：30000(30 秒)

deploymentConfigurationTimeSource

此參數適用於此元件的 v2.15.0 和更新版本。

(選用) 處理部署時要使用的時間戳記。預設值為 `deploymentCreationTime`。

此物件包含下列值：

deploymentCreationTime

預設值 `deploymentConfigurationTimeSource`。裝置使用部署建立時間戳記來解決處理期間的組態金鑰衝突。選取此行為時，核心所保留的本機裝置組態可能具有比傳入部署更大的時間戳記，並拒絕現在視為過期的傳入組態變更。

deploymentProcessingTime

裝置會使用其本機時間戳記來解決部署處理期間的組態金鑰衝突。處理時，裝置會根據處理時間戳記而非部署建立時間戳記來更新組態。此行為假設裝置時鐘已正確校正。

如果您希望新裝置在第一次連線時使用此行為，請在初始裝置映像或安裝中設定此核心設定，而不是透過部署進行設定。使用此組態的 nucleus 傳統安裝程式 [--init-config](#) 選項。

此初始組態至關重要，因為裝置會依任意順序處理多個部署。如果沒有適當的初始組態，裝置可能會在接收將核心組態設定為 的部署之前，使用預設 deploymentCreationTime 行為來處理部署 deploymentProcessingTime。

Example 範例：組態合併更新

```
{
  "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"
  }
}
```

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.16.1	錯誤修正和改善 新增 Token Exchange Service 失敗後登入資料重試間隔的組態。
2.16.0	新功能 - 現在，Nucleus 支援 cgroups V2。 錯誤修正和改善 - 修正當部署取消時，核心無法清除過時部署的問題。
2.15.1	錯誤修正和改善 修正引導部署期間 30 秒關閉逾時後，核心無法清除元件程序的問題。
2.15.0	新功能 - 新增遙測功能，以包含主機系統資訊，例如 CPU 和作業系統詳細資訊。如需詳細資訊，請參閱遙測元件頁面。 - 新增 deploymentConfigurationTimeSource 組態。如需詳細資訊，請參閱 核元件組態。

版本	變更
	錯誤修正和改善 • 根據預設，Nucleus 現在會優先使用本機版本和成品，從而提高部署一致性並減少外部相依性。使用者可以透過在部署文件中指定替代偏好設定來覆寫此行為。 • 當提供的 Windows 使用者不符合 Windows 通常接受的字元集時，新增警告日誌。
2.14.3	錯誤修正和改善 • 允許權杖交換服務在連接埠組態變更時重新啟動。 • 修正機群狀態服務未觸發未鎖定 Lambda 時，不會傳送元件狀態變更訊息的問題。 • 修正部署新元件版本時，元件無法正常關閉的問題。 • 修正內建外掛程式的注入掛鉤執行兩次導致額外的生命週期事件和重複記錄的問題。 • 改善 Windows 裝置上的元件生命週期記錄。
2.14.2	錯誤修正和改善 • 修正 HTTP 用戶端未設定交互身分驗證的問題。
2.14.1	錯誤修正和改善 • 修正 Greengrass 新安裝時元件未正確停止的問題。

版本	變更
2.14.0	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 新功能 - 新的雙堆疊端點支援可啟用 IPv6 網路通訊。 - 增強對核重新啟動失敗和Launch目錄損毀的彈性。 錯誤修正和改善 - 修正 IPC PubSub 訂閱關閉中的記憶體流失。 - 修正略過條件為 true 時，因啟動逾時而進入 ERRORED 狀態之元件的執行生命週期。 - 修正核心裝置在 TLS 政策設定為 TLS13_1_3_2022_10 時無法連線至 AWS IoT Core 的問題。
2.13.0	新功能 - 支援《Nucleus》中的 FIPS 端點。 錯誤修正和改善 - 取消部署改進：現在可在合併新組態時以及等待服務啟動時取消部署。
2.12.6	錯誤修正和改善 修正在特定 ARMv8 處理器上造成啟動時當機的問題，包括 Jetson Nano。
2.12.5	錯誤修正和改善 - 修正部署轉返偶爾在轉返具有硬相依性的先前損壞元件時卡住的問題。 - 修正機群佈建後，核不會發佈狀態更新的問題。 - 收到 404 錯誤後，新增 GetDeploymentConfiguration API 的重試次數。
2.12.4	錯誤修正和改善 修正在某些 Linux 裝置上啟動時，核心進入死鎖條件的問題。

版本	變更
2.12.3	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 - 修正核心在重新啟動後和元件復原期間未報告正確元件狀態的問題。 - 一般錯誤修正與改進。
2.12.2	錯誤修正和改善 - 修正舊日誌未正確清除的問題。 - 一般錯誤修正與改進。
2.12.1	錯誤修正和改善 修正核心可能將 MQTT 訂閱複製到部署主題導致額外記錄和 MQTT 發佈的問題。
2.12.0	新功能 可讓您在轉返部署中執行引導生命週期步驟。
2.11.3	錯誤修正和改善 - 修正核心中可能在元件相依性失敗時不當啟動元件的問題。 新功能 - 新增可設定的 s3 端點類型。
2.11.2	錯誤修正和改善 - 修正使用大量 (> 50) 訂閱時，可能離線的 nucleus MQTT 5 用戶端問題。 - 新增 Docker 撥號 TCP 失敗的重試。
2.11.1	錯誤修正和改善 - 修正當引導任務失敗且部署中繼資料檔案損毀時，未啟動核心的問題。 - 修正部署狀態更新中未報告隨需 Lambda 元件的問題。 - 新增對重複授權政策 IDs 支援。

版本	變更
2.11.0	新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的故障處理政策。 • 新增對磁碟多工緩衝處理外掛程式的支援。
2.10.3	錯誤修正和改善 • 修正 Greengrass 在使用 PKCS#11 供應商時未訂閱部署通知的問題。
2.10.2	錯誤修正和改善 • 允許不區分大小寫的元件生命週期剖析。 • 修正環境 PATH 變數未正確重新建立的問題。 • 修正元件的代理 URI 編碼，包括具有特殊字元之使用者名稱的串流管理員。
2.10.1	錯誤修正和改善 • 修正在特定 ARMv8 處理器上啟動時可能導致當機的問題，包括 Jetson Nano。 • Greengrass 不再關閉元件的標準，這會將行為還原為 pre-2.10.0 行為

版本	變更
2.10.0	新功能 • 新增對空白規則表達式的<code>interpolateComponentConfiguration</code> 支援。Greengrass 現在會從根組態物件插入。 • 新增對 MQTT5 的支援。 • 新增無需掃描即可快速載入外掛程式元件的機制。 • 讓 Greengrass 透過刪除未使用的 Docker 映像來節省磁碟空間。 錯誤修正和改善 • 修正轉返從部署保留特定組態值的問題。 • 修正 Greengrass 核在自訂非AWS 憑證和資料端點中驗證 AWS 網域序列的問題。 • 更新多群組相依性解析，透過 AWS 雲端 交涉重新解析所有群組相依性，而不是鎖定作用中的版本。此更新也會移除部署錯誤碼 <code>INSTALLED_COMPONENT_NOT_FOUND</code> 。 • 更新 Greengrass 核，以便在本機已存在時略過下載 Docker 映像。 • 在逾時到期之前，更新 Greengrass 核以重新啟動元件安裝步驟。 • 其他次要修正和改進。
2.9.6	錯誤修正和改善 • 修正 Greengrass 部署失敗並顯示 <code>LAUNCH_DIRECTORY_CORRUPTED</code> 錯誤，且後續裝置重新啟動無法啟動 Greengrass 的問題。當您在部署需要 Greengrass 重新啟動的多個物件群組之間移動 Greengrass 裝置時，可能會發生此錯誤。
2.9.5	新功能 • 新增對 Greengrass 核軟體簽章驗證的支援。 錯誤修正和改善 • 修正當本機配方中繼資料區域與 Greengrass 核啟動區域不相符時，部署失敗的問題。發生這種情況時，Greengrass 核現在會與雲端重新交涉。 • 修正 MQTT 訊息多工緩衝處理器填滿且永遠不會移除訊息的問題。 • 其他次要修正和改進。

版本	變更
2.9.4	錯誤修正和改善 - 在捨棄 QOS 0 訊息之前檢查 null 訊息。 - 如果任務狀態詳細資訊值超過 1024 個字元限制，則會將其截斷。 - 如果該路徑包含空格，請更新 Windows 的引導指令碼，以正確讀取 Greengrass 根路徑。 - 更新訂閱，AWS IoT Core 以便在未傳送訂閱回應時捨棄用戶端訊息。 - 確保當主要組態檔案損毀或遺失時，核會從備份檔案載入其組態。
2.9.3	錯誤修正和改善 - 確保 MQTT IDs 不會重複。 - 新增更強大的檔案讀取和寫入，以避免並從損毀中復原。 - 重試 Docker 映像提取特定網路相關錯誤。 - 新增 MQTT 連線noProxyAddresses 的選項。
2.9.2	錯誤修正和改善 - 修正設定 interpolateComponentConfiguration 不適用於進行中部署的問題。 - 使用 OSHI 列出所有子程序。
2.9.1	錯誤修正和改善 當部署移除外掛程式元件時，新增 Greengrass 重新啟動的修正。
2.9.0	新功能 - 新增建立子部署的功能，以使用較小的裝置子集重試部署。此功能會建立更有效率的方式來測試和解決失敗的部署。 錯誤修正和改善 - 改善對沒有 useradd、groupadd和 之系統的支援usermod。 - 其他次要修正和改進。

版本	變更
2.8.1	錯誤修正和改善 - 修正 Greengrass API 錯誤未正確產生部署錯誤代碼的問題。 - 修正當元件在部署期間達到 ERRORED 狀態時，機群狀態更新會傳送不正確資訊的問題。 - 修正 Greengrass 有超過 50 個現有訂閱時，部署無法完成的問題。
2.8.0	新功能 - 更新 Greengrass 核以報告部署運作狀態回應，其中包含在將元件部署至核心裝置時的詳細錯誤代碼。如需詳細資訊，請參閱詳細部署錯誤代碼。 - 更新 Greengrass 核以報告元件運作狀態回應，其中包含元件進入 BROKEN 或 ERRORED 狀態時的詳細錯誤代碼。如需詳細資訊，請參閱詳細的元件狀態碼。 - 展開狀態訊息欄位，以改善裝置的雲端可用性資訊。 - 改善機群狀態服務穩健性。 錯誤修正和改善 - 允許損壞的元件在其組態變更時重新安裝。 - 修正在引導部署期間重新啟動核心會導致部署失敗的問題。 - 修正根路徑包含空格時安裝失敗的 Windows 問題。 - 修正元件在部署期間關閉使用新版本的關閉指令碼的問題。 - 各種關機改進。 - 其他次要修正和改進。

版本	變更
2.7.0	新功能 - 當核心裝置套用本機部署時，更新 Greengrass 核以傳送狀態更新至 AWS IoT Greengrass 雲端。 - 新增由自訂憑證授權單位 (CA) 簽署的用戶端憑證支援，其中 CA 未註冊 AWS IoT。若要使用此功能，您可以將新的greengrassDataPlaneEndpoint 組態選項設定為 iotdata。如需詳細資訊，請參閱使用私有 CA 簽署的裝置憑證。 錯誤修正和改善 - 修正在特定情況下，Greengrass 核停止或重新啟動時復原部署的問題。核現在會在核重新啟動後繼續部署。 - 當您指定 將軟體設定為系統服務時，請更新 Greengrass 安裝程式以遵守 --start 引數。 - 更新的行為SubscribeToComponentUpdates，以在核心更新元件的事件中設定部署 ID。 - 其他次要修正和改進。

版本	變更
2.6.0	新功能 - 當您訂閱本機發佈/訂閱主題時，新增對 MQTT 萬用字元的支援。如需詳細資訊，請參閱發佈/訂閱本機訊息及SubscribeToTopic。 - 新增對元件組態中配方變數的支援，配方<code>component_dependency_name</code>:configuration: <code>json_pointer</code> 變數除外。您可以在配方DefaultConfiguration 中定義元件的 或設定部署中的元件時，使用這些配方變數。若要啟用此功能，請將 interpolateComponentConfiguration 組態選項設定為 true。如需詳細資訊，請參閱配方變數及在合併更新中使用配方變數。 - 新增對程序間通訊 (IPC) 授權政策中*萬用字元的完整支援。您現在可以在資源字串中指定*字元，以符合字元的任意組合。如需詳細資訊，請參閱授權政策中的萬用字元。 - 新增對自訂元件的支援，以呼叫 Greengrass CLI 使用的 IPC 操作。您可以使用這些 IPC 操作來管理本機部署、檢視元件詳細資訊，以及產生可用來登入本機偵錯主控台的密碼。如需詳細資訊，請參閱 IPC：管理本機部署和元件。 錯誤修正和改善 - 修正相依元件在特定情況下重新啟動或變更狀態時無法反應的問題。 - 改善核心裝置在部署失敗時向 AWS IoT Greengrass 雲端服務報告的錯誤訊息。 - 修正 Greengrass 核在特定案例重新啟動時套用物件部署兩次的問題。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。
2.5.6	新功能 - 新增對使用 ECC 金鑰之硬體安全模組的支援。您可以使用硬體安全模組 (HSM) 安全地存放裝置的私有金鑰和憑證。如需詳細資訊，請參閱硬體安全整合。 錯誤修正和改善 - 修正在特定案例中使用中斷的安裝指令碼部署元件時，部署從未完成的問題。 - 改善啟動期間的效能。 - 其他次要修正和改進。

版本	變更
2.5.5	新功能 - 新增元件GG_ROOT_CA_PATH 的環境變數，讓您可以存取自訂元件中的根憑證授權單位 (CA) 憑證。 錯誤修正和改善 - 新增對使用英文以外顯示語言之 Windows 裝置的支援。 - 更新 Greengrass 核如何剖析布林值安裝程式引數，因此您可以指定不含布林值的布林值引數來指定true值。例如，您現在可以指定 --provision true ，--provision 而不是使用自動資源佈建來安裝。 - 修正核心裝置在特定情況下佈建後未向 AWS IoT Greengrass 雲端服務報告其狀態的問題。 - 其他次要修正和改進。
2.5.4	錯誤修正和改善 一般錯誤修正與改進。
2.5.3	新功能 - 新增硬體安全整合的支援。您可以使用硬體安全模組 (HSM) 安全地存放裝置的私有金鑰和憑證。如需詳細資訊，請參閱硬體安全整合。 錯誤修正和改善 - 修正當核心與 建立 MQTT 連線時，執行時間例外狀況的問題 AWS IoT Core。
2.5.2	錯誤修正和改善 修正 Greengrass 核更新後，Windows 服務在您停止或重新啟動裝置後無法再次啟動的問題。

版本	變更
2.5.1	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 • 新增對 Windows 上 32 位元版本的 Java 執行期環境 (JRE) 的支援。 • 變更政策 AWS IoT 未授予greengrass:ListThingGroupsForCoreDevice 許可之核心裝置的物件群組移除行為。在此版本中，部署會繼續、記錄警告，而且當您從物件群組移除核心裝置時，不會移除元件。如需詳細資訊，請參閱將 AWS IoT Greengrass 元件部署至裝置。 • 修正 Greengrass 核提供給 Greengrass 元件程序的系統環境變數問題。您現在可以重新啟動元件以使用最新的系統環境變數。

版本	變更
2.5.0	新功能 • 新增對執行 Windows 之核心裝置的支援。 • 變更物件群組移除的行為。使用此版本，您可以從物件群組中移除核心裝置，以在下次部署中解除安裝該物件群組的元件。 由於此變更，核心裝置 AWS IoT 的政策必須具有 <code>greengrass::ListThingGroupsForCoreDevice</code> 許可。如果您使用 AWS IoT Greengrass Core 軟體安裝程式佈建資源，預設 AWS IoT 政策會允許 <code>greengrass:*</code>，其中包含此許可。如需詳細資訊，請參閱裝置身分驗證和授權 AWS IoT Greengrass。 • 新增對 HTTPS 代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理。 • 新增新的 <code>windowsUser</code> 組態參數。您可以使用此參數來指定預設使用者，以用來在 Windows 核心裝置上執行元件。如需詳細資訊，請參閱設定執行元件的使用者。 • 新增可用於自訂 HTTP 請求逾時的新 <code>httpClient</code> 組態選項，以改善慢速網路的效能。如需詳細資訊，請參閱 httpClient 組態參數。 錯誤修正和改善 • 修正引導生命週期選項，以從元件重新啟動核心裝置。 • 在配方變數中新增連字號的支援。 • 修正隨需 Lambda 函數元件的 IPC 授權。 • 改善日誌訊息，並將非關鍵日誌從 INFO 變更為 DEBUG 關卡，因此日誌更實用。 • 從 Greengrass 核在安裝具有自動佈建 AWS IoT Greengrass 的核心軟體時建立的預設權杖交換角色中移除 <code>iot:DescribeCertificate</code> 許可。Greengrass 核不會使用此許可。 • 修正問題，以便在 <code>iam:CreatePolicy</code> 適用於相同政策時，自動佈建指令碼不需要 <code>iam:GetPolicy</code> 許可。 • 其他次要修正和改進。

版本	變更
2.4.0	新功能 • 新增對系統資源限制的支援。您可以設定每個元件程序可在核心裝置上使用的 CPU 和 RAM 用量上限。如需詳細資訊，請參閱設定元件的系統資源限制。 • 新增 IPC 操作以暫停和繼續元件。如需詳細資訊，請參閱PauseComponent及ResumeComponent。 • 新增佈建外掛程式的支援。您可以指定在安裝期間執行的 JAR 檔案，以佈建 Greengrass 核心裝置所需的 AWS 資源。Greengrass 核包含一個介面，您可以實作此介面來開發自訂佈建外掛程式。如需詳細資訊，請參閱使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體。 • 將選用thing-name-policy 引數新增至 AWS IoT Greengrass Core 軟體安裝程式。當您使用自動資源佈建安裝 AWS IoT Greengrass Core 軟體時，可以使用此選項指定現有或自訂 AWS IoT 政策。 錯誤修正和改善 • 更新啟動時的記錄組態。這修正了啟動時未套用記錄組態的問題。 • 更新 nucleus 載入器符號連結，以在安裝期間指向 Greengrass 根資料夾中的元件存放區。此更新可讓您刪除安裝 AWS IoT Greengrass 核心軟體時下載的 JAR 檔案和其他核子成品。 • 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。

版本	變更
2.3.0	新功能 - 新增對部署組態文件的支援，最高可達 10 MB，從 7 KB（適用於以物件為目標的部署）或 31 KB（適用於以物件群組為目標的部署）開始。 若要使用此功能，核心裝置的 AWS IoT 政策必須允許 <code>greengrass:GetDeploymentConfiguration</code> 許可。如果您使用 AWS IoT Greengrass Core 軟體安裝程式佈建資源，則核心裝置的 AWS IoT 政策會允許 <code>greengrass:*</code>，其中包含此許可。如需詳細資訊，請參閱裝置身分驗證和授權 AWS IoT Greengrass。 - 新增 <code>iot:thingName</code> 配方變數。您可以使用此配方變數來取得配方中核心裝置 AWS IoT 物件的名稱。如需詳細資訊，請參閱配方變數。 錯誤修正和改善 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。
2.2.0	新功能 - 新增本機影子管理的 IPC 操作。 錯誤修正和改善 - 減少 JAR 檔案的大小。 - 減少記憶體用量。 - 修正在某些情況下未更新日誌組態的問題。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。

版本	變更
2.1.0	新功能 • 支援從 Amazon ECR 中的私有儲存庫下載 Docker 映像。 • 新增下列參數來自訂核心裝置上的 MQTT 組態： • <code>maxInFlightPublishes</code> – 可同時傳輸的未確認 MQTT QoS 1 訊息數量上限。 • <code>maxPublishRetry</code> – 重試無法發佈訊息的次數上限。 • 新增 <code>fleetstatusservice</code> 組態參數，以設定核心裝置將裝置狀態發佈至的間隔 AWS 雲端。 • 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。 錯誤修正和改善 • 修正核重新啟動時導致陰影部署重複的問題。 • 修正當發生服務載入例外狀況時，導致核心當機的問題。 • 改善元件相依性解析，使包含循環相依性的部署失敗。 • 修正如果先前已從核心裝置移除外掛程式元件，導致該元件無法重新部署的問題。 • 修正導致 HOME 環境變數設定為 Lambda 元件或執行為根的元件 / <i>greengrass/v2</i> /work 目錄的問題。HOME 變數現在已正確設定為執行元件之使用者的主目錄。 • 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的版本。
2.0.5	錯誤修正和改善 • 下載 AWS 提供的元件時，透過設定的網路代理正確路由流量。 • 在中國區域使用正確的 Greengrass 資料平面端點 AWS。

版本	變更
2.0.4	新功能 - 啟用透過連接埠 443 的 HTTPS 流量。您可以使用 2.0.4 版 nucleus 元件的新 <code>greengrassDataPlanePort</code> 組態參數，將 HTTPS 通訊設定為透過連接埠 443 傳輸，而不是預設連接埠 8443。如需詳細資訊，請參閱 透過連接埠 443 設定 HTTPS。 - 新增工作路徑配方變數。您可以使用此配方變數來取得元件工作資料夾的路徑，可用來在元件及其相依性之間共用檔案。如需詳細資訊，請參閱 工作路徑配方變數。 錯誤修正和改善 - 如果角色政策已存在，則防止建立字交換 AWS Identity and Access Management (IAM) 角色政策。 由於此變更，安裝程式現在在使用執行 <code>sts:GetCallerIdentity</code> 時需要 <code>iam:GetPolicy</code> 和 <code>--provision true</code>。如需詳細資訊，請參閱 安裝程式佈建資源的最低 IAM 政策。 - 正確處理尚未成功註冊的部署取消。 - 更新組態，以在復原部署時移除具有較新時間戳記的較舊項目。 - 其他次要修正和改進。如需詳細資訊，請參閱 GitHub 上的 版本。
2.0.3	初始版本。

Greengrass nucleus lite

Greengrass nucleus lite (`aws.greengrass.NucleusLite`) 是針對限制邊緣裝置的裝置執行時間，針對最少的記憶體使用量（使用少於 5MB 的 RAM）進行最佳化。它隨 2.14.0 AWS IoT Greengrass 版推出，並且與 AWS IoT Greengrass 一般元件、Greengrass V2 API 和 SDK 回溯相容。

Greengrass 核精簡版是通用 [Greengrass 核 \(`aws.greengrass.Nucleus`\)](#) 的替代方案，可用於 Greengrass 裝置的異質機群。

主題

- [版本](#)
- [作業系統](#)

- [要求](#)
- [相容性](#)
- [下載並安裝](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux（使用 systemd 的分佈）

如需詳細資訊，請參閱 [Greengrass 核](#)。

要求

裝置必須符合特定需求，才能安裝和執行 nucleus lite AWS IoT Greengrass 和 AWS IoT Greengrass Core 軟體。如需詳細資訊，請參閱[設定指南](#)。

- 核執行時間的 5MB RAM 空間。
- 5MB 的儲存體（磁碟/快閃記憶體）。

其他系統相依性記錄在 [設定指南](#) 中。

Greengrass 核元件支援在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目：

- Greengrass 核必須能夠連線至 AWS IoT 資料、AWS IoT 憑證和 Amazon S3。

相容性

AWS IoT Greengrass nucleus lite 與 AWS IoT Greengrass v2 API（子集）和支援的 SDKs 相容。它不依賴任何特定語言執行時間/VMs但新增至部署的元件可能需要特定執行時間（例如：Java JVM、Python）。如需 Greengrass nucleus lite 支援哪些功能的詳細資訊，請參閱 [Greengrass 功能相容性](#)。

下載並安裝

您可以下載 apt 套件、[從來源建置](#)、[使用 Yocto layer](#)，或[下載相容裝置（例如 RaspberryPi）的預先建置 Yocto 映像](#)。RaspberryPi 從 [AWS IoT Core 主控台](#)，您將能夠下載包含裝置所有登入資料和初始組態的連線套件。如何安裝的指示會包含在每個特定的分發方法中。

您也可以遵循教學課程來安裝 nucleus lite AWS IoT Greengrass 並探索 Greengrass 元件開發。如需詳細資訊，請參閱[教學課程：入門 AWS IoT Greengrass V2](#)。

Configuration

核提供下列[組態](#)參數。有些參數需要 AWS IoT Greengrass Core 軟體重新啟動才能生效。

iotRoleAlias

指向字符交換 IAM AWS IoT 角色的角色別名。AWS IoT 登入資料提供者會擔任此角色，以允許 Greengrass 核心裝置與 AWS 服務互動。如需詳細資訊，請參閱[授權核心裝置以與服務互動 AWS](#)。

iotDataEndpoint

您 AWS IoT 的資料端點 AWS 帳戶。

iotCredEndpoint

您的 AWS IoT 登入資料端點 AWS 帳戶。

greengrassDataPlanePort

用於資料平面連線的連接埠。如需詳細資訊，請參閱在[連接埠 443 上或透過網路代理連線](#)。

Important

您必須指定裝置可以進行傳出連線的連接埠。如果您指定封鎖的連接埠，裝置將無法連線至 AWS IoT Greengrass 以接收部署。您可以從以下選項中選擇：

- 443

- 8443
- 預設：8443

awsRegion

AWS 區域 要使用的。

runWithDefault

用來執行元件的系統使用者。

Important

當您部署變更至此組態參數時，AWS IoT Greengrass 核心軟體會重新啟動，讓變更生效。

此物件包含下列資訊：

posixUser

系統使用者的名稱或 ID，也可以選擇核心裝置用來執行一般元件的系統群組。以下列格式指定使用者和群組，並以冒號 (:) 分隔：user:group。群組為選用項目。如果您未指定群組，則 AWS IoT Greengrass Core 軟體會使用使用者的主要群組。例如，您可以指定 ggc_user 或 ggc_user:ggc_group。如需詳細資訊，請參閱[設定執行元件的使用者](#)。

networkProxy

(選用) 用於所有連線的網路代理。如需詳細資訊，請參閱[連線至連接埠 443 或透過網路代理](#)。

Important

當您部署變更至此組態參數時，變更會在下一次重新啟動 AWS IoT Greengrass Core 軟體後生效。

此物件包含下列資訊：

noProxyAddresses

(選用) 從代理中豁免的 IP 地址或主機名稱的逗號分隔清單。

proxy

要連線的代理。此物件包含下列資訊：

url

代理伺服器的 URL，格式為 `http://host:port`。

- `scheme` – 結構描述，必須是 `http`。
- `host` – 代理伺服器的主機名稱或 IP 地址。
- `port` – （選用）連接埠號碼。如果您未指定連接埠，則 Greengrass 核心裝置會使用下列預設值：
 - `http` – 80

本機日誌檔案

訊息會記錄到 `stdout`，日誌檔案由系統化處理。

檢視此元件的日誌

- 使用 `journalctl` 來檢視日誌。

變更記錄

版本	變更
2.3.2	錯誤修正和改善 • 更新版本檔案，以取得正確的 <code>nucleus</code> 版本報告。
2.3.1	錯誤修正和改善 • GG 不會嘗試更新已取消任務的部署狀態。 • 確定 <code>fdopendir</code> 的 <code>DIR</code> 已關閉。 • 機群佈建現在會在每次執行時觸發憑證檔案覆寫。 • 其他次要錯誤修正。
2.3.0	新功能 • 支援將 TPM 2.0 用於 IoT Core MQTT 授權。

版本	變更
	- 範例 apt 套件現在支援更多作業系統：Ubuntu 22.04、Ubuntu 24.04、Debian 12 和 Debian 13。 - 現在支援 RestartComponent IPC。 錯誤修正和改善 - 本機部署不再需要網際網路存取。 - GetConfiguration 已更新，以符合 Greengrass Nucleus 執行時間行為。(重大變更) - 一般錯誤修正與改進。
2.2.2	錯誤修正和改善 - 修正在未變更元件具有執行中可執行檔做為成品時，包含未變更元件版本的修訂部署失敗。 - 當插入值具有四個以上的巢狀子物件時，修正配方變數插補。 - 修正配方變數插補，包括引號和 Shell 特殊字元。 - 修正 MQTT 連線失敗時的 fds 洩漏問題。
2.2.1	錯誤修正和改善 修正核心無法取得 TES 登入資料的問題。
2.2.0	新功能 - 新增對容器映像成品 URIs 支援。 錯誤修正和改善 - 一般錯誤修正與改進。

版本	變更
2.1.0	新功能 - 新增可使用networkProxy 組態選項設定的 HTTP 代理支援。 錯誤修正和改善 - 將的需求libcurl從 7.86 降低為 7.82，以容納執行較舊libcurl版本的裝置。 - 更新journalctl 日誌，使其歸因於元件而非配方執行器。 - 改善 IPC 呼叫的錯誤回應。 - 針對一般元件成品新增 S3 下載嘗試的重試。 - 次要錯誤修正。
2.0.2	錯誤修正和改善 修正 apt 套件的相依性，以包含 cgroup-tools 。
2.0.1	錯誤修正和改善 - 新增 Greengrass 配方逾時區段的配方變數插補支援。 - 新增對串流管理員的 ValidateAuthorizationToken IPC 命令的支援。 - 修正機群佈建的警告。 - 將重試和退避新增至任務接聽程式。 - 一般錯誤修正與改進。
2.0.0	初始版本。

用戶端裝置身分驗證

用戶端裝置驗證元件 (aws.greengrass.clientdevices.Auth) 會驗證用戶端裝置，並授權用戶端裝置動作。

Note

用戶端裝置是連線至 Greengrass 核心裝置的本機 IoT 裝置，用於傳送 MQTT 訊息和要處理的資料。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

Note

用戶端裝置驗證 2.3.0 版已停止。我們強烈建議您升級至用戶端裝置驗證版本 2.3.1 或更新版本。

此元件具有下列版本：

- 2.5.x
- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- [Greengrass 服務角色](#)必須與您的 相關聯，AWS 帳戶 並允許 `iot:DescribeCertificate`許可。
- 核心裝置的 AWS IoT 政策必須允許下列許可：
 - `greengrass:GetConnectivityInfo`，其中資源包含執行此元件之核心裝置的 ARN
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`，其中資源包含連接至核心裝置之每個用戶端裝置的 Amazon Resource Name (ARN)
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:PutCertificateAuthorities`
 - `iot:Publish`，其中資源包含下列 MQTT 主題的 ARN：
 - `$aws/things/coreDeviceThingName*-gci/shadow/get`
 - `iot:Subscribe`，其中資源包含下列 MQTT 主題篩選條件的 ARNs：
 - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
 - `iot:Receive`，其中資源包含下列 MQTT 主題 ARNs：
 - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
 - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

如需詳細資訊，請參閱[AWS IoT 資料平面操作的 政策](#)及[支援用戶端裝置的最低 AWS IoT 政策](#)。

- (選用) 若要使用離線身分驗證，服務使用的 AWS Identity and Access Management AWS IoT Greengrass (IAM) 角色必須包含下列許可：
 - `greengrass:ListClientDevicesAssociatedWithCoreDevice` 讓核心裝置列出用戶端以進行離線身分驗證。
- 用戶端裝置身分驗證元件支援在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。

- 用戶端裝置身分驗證元件必須連線至 AWS IoT data AWS IoT 、登入資料和 Amazon S3。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	連線埠	必要	說明
iot. <i>region</i> .amazonaws.com	443	是	用來取得 AWS IoT 物件憑證的相關資訊。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.5.5

下表列出此元件 2.5.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.17.0	軟式

2.5.4

下表列出此元件 2.5.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.16.0	軟式

2.5.2 – 2.5.3

下表列出此元件 2.5.2 和 2.5.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.15.0	軟式

2.5.1

下表列出此元件 2.5.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.14.0	軟式

2.4.4 - 2.5.0

下表列出此元件 2.4.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.13.0	軟式

2.4.3

下表列出此元件 2.4.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.12.0	軟式

2.4.1 and 2.4.2

下表列出此元件 2.4.1 和 2.4.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.11.0	軟式

2.3.0 – 2.4.0

下表列出此元件 2.3.0 至 2.4.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.10.0	軟式

2.3.0

下表列出此元件 2.3.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.10.0	軟式

2.2.3

下表列出此元件 2.2.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <=2.9.0	軟式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <=2.8.0	軟式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.8.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.6.0 <2.7.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.7.0	軟式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.6.0	軟式

2.0.2 and 2.0.3

下表列出此元件 2.0.2 和 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.5.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.4.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.3.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

Note

在向本機 MQTT 代理程式提出用戶端訂閱請求時，會評估訂閱許可。如果用戶端的現有訂閱許可遭到撤銷，用戶端將無法再訂閱主題。不過，它會繼續接收來自任何先前訂閱主題的訊息。為了防止這種行為，應在撤銷訂閱許可可以強制重新授權用戶端之後重新啟動本機 MQTT 代理程式。

對於 MQTT 5 代理程式 (EMQX) 元件，請更新restartIdentifier組態以重新啟動 MQTT 5 代理程式。

對於 MQTT 3.1.1 代理程式 (Moquette) 元件，當伺服器憑證變更強制用戶端重新授權時，預設會每週重新啟動。您可以變更核心裝置的連線資訊 (IP 地址) 或進行部署以移除代理程式元件，然後稍後再次部署，強制重新啟動。

v2.5.0 – 2.5.4

deviceGroups

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義用戶端裝置授權政策，以指定每個裝置群組的許可。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則至少包含一個選擇規則子句，這是單一表達式查詢，可以符合用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選擇規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱的開頭和結尾使用此萬用字元，來比對名稱開頭或結尾為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

 Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\:ClientDevice1` 選取名稱為 的物件 `MyTeam:ClientDevice1`。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名為 `MyClientDevice1` 或 的用戶端裝置 `MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 的用戶端裝置 `MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱結尾為 的用戶端裝置 `MyClientDevice`。

```
thingName: *MyClientDevice
```

Example 範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

policyName

套用至此裝置群組中用戶端裝置的許可政策。指定您在 policies 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:deviceClientId` – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:mqttTopic` – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:`*mqttTopicFilter* – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源不支援 MQTT 主題萬用字元。

resources

允許此政策中操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:`*mqttTopic*) 的清單。

您可以在資源變數內的任何位置指定 * 萬用字元，以允許存取所有資源。例如，您可以指定 `mqtt:topic:my*` 以允許存取符合該輸入的資源。

支援下列資源變數：

- `mqtt:topic:${iot:Connection.Thing.ThingName}`

這會解析為正在評估政策之 AWS IoT Core 登錄檔中的物件名稱。AWS IoT Core 會使用裝置在驗證時提供的憑證來判斷要使用哪些物件來驗證連線。本政策變數僅在裝置透過 MQTT 或 MQTT over WebSocket 通訊協定進行連線時可用。

statementDescription

(選用) 此政策陳述式的描述。

certificates

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

serverCertificateValiditySeconds

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況

時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000 (10 天)

performance

(選用) 此核心裝置的效能組態選項。此物件包含下列資訊：

maxActiveAuthTokens

(選用) 作用中用戶端裝置授權字符的數量上限。您可以增加此數量，讓更多用戶端裝置連接到單一核心裝置，而無需重新驗證它們。

預設：2500

cloudRequestQueueSize

(選用) 此元件拒絕 AWS 雲端 請求之前佇列的請求數目上限。

預設：100

maxConcurrentCloudRequests

(選用) 要傳送至 的並行請求數目上限 AWS 雲端。您可以增加此數字，以改善連接大量用戶端裝置的核心裝置上的身分驗證效能。

預設：1

certificateAuthority

(選用) 憑證授權單位組態選項，可將核心裝置中繼授權單位取代為您自己的中繼憑證授權單位。

Note

如果您使用自訂憑證授權機構 (CA) 設定 Greengrass 核心裝置，並使用相同的 CA 發出用戶端裝置憑證，Greengrass 會略過用戶端裝置 MQTT 操作的授權政策檢查。用戶端裝置身分驗證元件會使用其設定為使用的 CA 簽署的憑證，完全信任用戶端。

若要在使用自訂 CA 時限制此行為，請使用不同的 CA 或中繼 CA 建立和簽署用戶端裝置，然後調整 `certificateUri` 和 `certificateChainUri` 欄位以指向正確的中繼 CA。

此物件包含下列資訊。

`certificateUri`

憑證的位置。它可以是檔案系統 URI 或指向存放在硬體安全模組中的憑證的 URI。

`certificateChainUri`

核心裝置 CA 的憑證鏈位置。這應該是返回根 CA 的完整憑證鏈。它可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證鏈。

`privateKeyUri`

核心裝置私有金鑰的位置。這可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證私有金鑰。

`security`

(選用) 此核心裝置的安全組態選項。此物件包含下列資訊。

`clientDeviceTrustDurationMinutes`

在使用核心裝置重新驗證之前，用戶端裝置的身分驗證資訊可信任的持續時間，以分鐘為單位。預設值為 1。

`metrics`

(選用) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

`disableMetrics`

如果 `disableMetrics` 欄位設定為 `true`，則用戶端裝置身分驗證不會收集指標。

預設：`false`

`aggregatePeriodSeconds`

以秒為單位的彙總期間，決定用戶端裝置身分驗證彙總指標並將其傳送至遙測代理程式的頻率。這不會變更指標發佈的頻率，因為遙測代理器仍會每天發佈一次。

預設：3600

startupTimeoutSeconds

(選用) 元件啟動的秒數上限。ERRORED 如果元件超過此逾時，元件的狀態會變更為。

預設：120

Example範例：組態合併更新（使用限制性政策）

下列範例組態指定，允許名稱開頭為 的用戶端裝置MyClientDevice在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```

{
  "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 範例：組態合併更新（使用物件名稱政策）

下列範例組態可讓用戶端裝置在以用戶端裝置物件名稱開頭並以字串結尾的主題上發佈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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義指定每個裝置群組許可的用戶端裝置授權政策。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則至少包含一個選擇規則子句，這是單一表達式查詢，可以符合用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱的開頭和結尾使用此萬用字元，來比對名稱開頭或結尾為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 的用戶端裝置MyClientDevice。

```
thingName: MyClientDevice*
```

Example範例選取規則（使用萬用字元）

下列選擇規則符合名稱以 結尾的用戶端裝置MyClientDevice。

```
thingName: *MyClientDevice
```

Example範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

policyName

套用至此裝置群組中用戶端裝置的許可政策。指定您在 policies 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:deviceClientId` – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:mqttTopic` – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:mqttTopicFilter` – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 + 和 # MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 * 萬用字元，以允許存取所有動作。

resources

此政策中允許 操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:mqttTopic`) 的清單。

您可以指定 * 萬用字元，以允許存取所有資源。您無法使用 * 萬用字元來比對部分資源識別符。例如，您可以指定 **"resources": "*"，但無法指定 **"resources": "mqtt:clientId:*"****。

statementDescription

(選用) 此政策陳述式的描述。

certificates

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

serverCertificateValiditySeconds

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線和重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000 (10 天)

performance

(選用) 此核心裝置的效能組態選項。此物件包含下列資訊：

maxActiveAuthTokens

(選用) 作用中用戶端裝置授權字符的數量上限。您可以增加此數量，讓更多用戶端裝置連接到單一核心裝置，而無需重新驗證它們。

預設：2500

cloudRequestQueueSize

(選用) 此元件拒絕 AWS 雲端 請求之前佇列的請求數目上限。

預設：100

maxConcurrentCloudRequests

(選用) 要傳送至 的並行請求數目上限 AWS 雲端。您可以增加此數字，以改善連接大量用戶端裝置的核心裝置上的身分驗證效能。

預設：1

certificateAuthority

(選用) 憑證授權單位組態選項，可將核心裝置中繼授權單位取代為您自己的中繼憑證授權單位。

Note

如果您使用自訂憑證授權機構 (CA) 設定 Greengrass 核心裝置，並使用相同的 CA 發出用戶端裝置憑證，Greengrass 會略過用戶端裝置 MQTT 操作的授權政策檢查。用戶端裝置身分驗證元件會使用其設定為使用的 CA 簽署的憑證，完全信任用戶端。若要在使用自訂 CA 時限制此行為，請使用不同的 CA 或中繼 CA 建立和簽署用戶端裝置，然後調整 `certificateUri` 和 `certificateChainUri` 欄位以指向正確的中繼 CA。

此物件包含下列資訊。

certificateUri

憑證的位置。它可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證。

certificateChainUri

核心裝置 CA 的憑證鏈位置。這應該是返回根 CA 的完整憑證鏈。它可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證鏈。

privateKeyUri

核心裝置私有金鑰的位置。這可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證私有金鑰。

security

(選用) 此核心裝置的安全組態選項。此物件包含下列資訊。

clientDeviceTrustDurationMinutes

在使用核心裝置重新驗證之前，用戶端裝置的身分驗證資訊可信任的持續時間，以分鐘為單位。預設值為 1。

metrics

(選用) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

disableMetrics

如果 `disableMetrics` 欄位設定為 `true`，則用戶端裝置身分驗證不會收集指標。

預設：false

aggregatePeriodSeconds

以秒為單位的彙總期間，決定用戶端裝置身分驗證彙總指標並將其傳送至遙測代理程式的頻率。這不會變更指標發佈的頻率，因為遙測代理程式仍會每天發佈一次。

預設：3600

startupTimeoutSeconds

(選用) 元件啟動的時間上限，以秒為單位。ERRORED 如果元件超過此逾時，元件的狀態會變更為。

預設：120

Example範例：組態合併更新（使用限制性政策）

下列範例組態指定，允許名稱開頭為 `MyClientDevice` 的用戶端裝置在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```

{
    "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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義用戶端裝置授權政策，以指定每個裝置群組的許可。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則都包含至少一個選擇規則子句，這是單一表達式查詢，可以比對用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱結尾使用此萬用字元來比對名稱開頭為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \\。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\\\\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名稱為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 的用戶端裝置 MyClientDevice。

```
thingName: MyClientDevice*
```

Example 範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

policyName

套用至此裝置群組中用戶端裝置的許可政策。指定您在 policies 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:`*deviceClientId* – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:`*mqttTopic* – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:`*mqttTopicFilter* – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 + 和 # MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 * 萬用字元，以允許存取所有動作。

`resources`

允許此政策中操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:`*mqttTopic*) 的清單。

您可以指定 * 萬用字元，以允許存取所有資源。您無法使用 * 萬用字元來比對部分資源識別符。例如，您可以指定 `"resources": "*"` ，但無法指定 `"resources": "mqtt:clientId:*"`。

`statementDescription`

(選用) 此政策陳述式的描述。

certificates

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

serverCertificateValiditySeconds

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000 (10 天)

performance

(選用) 此核心裝置的效能組態選項。此物件包含下列資訊：

maxActiveAuthTokens

(選用) 作用中用戶端裝置授權字符的數量上限。您可以增加此數量，讓更多用戶端裝置連接到單一核心裝置，而無需重新驗證它們。

預設：2500

cloudRequestQueueSize

(選用) 此元件拒絕 AWS 雲端 請求之前佇列的請求數目上限。

預設：100

maxConcurrentCloudRequests

(選用) 要傳送至 的並行請求數目上限 AWS 雲端。您可以增加此數字，以改善連接大量用戶端裝置的核心裝置上的身分驗證效能。

預設：1

certificateAuthority

(選用) 憑證授權單位組態選項，可將核心裝置中繼授權單位取代為您自己的中繼憑證授權單位。

 Note

如果您使用自訂憑證授權機構 (CA) 設定 Greengrass 核心裝置，並使用相同的 CA 發出用戶端裝置憑證，Greengrass 會略過用戶端裝置 MQTT 操作的授權政策檢查。用戶端裝置身分驗證元件會使用其設定為使用的 CA 簽署的憑證，完全信任用戶端。若要在使用自訂 CA 時限制此行為，請使用不同的 CA 或中繼 CA 建立和簽署用戶端裝置，然後調整 `certificateUri` 和 `certificateChainUri` 欄位以指向正確的中繼 CA。

此物件包含下列資訊。

`certificateUri`

憑證的位置。它可以是檔案系統 URI 或指向存放在硬體安全模組中的憑證的 URI。

`certificateChainUri`

核心裝置 CA 的憑證鏈位置。這應該是返回根 CA 的完整憑證鏈。它可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證鏈。

`privateKeyUri`

核心裝置私有金鑰的位置。這可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證私有金鑰。

`security`

(選用) 此核心裝置的安全組態選項。此物件包含下列資訊。

`clientDeviceTrustDurationMinutes`

在使用核心裝置重新驗證之前，用戶端裝置的身分驗證資訊可信任的持續時間，以分鐘為單位。預設值為 1。

`metrics`

(選用) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

`disableMetrics`

如果 `disableMetrics` 欄位設定為 `true`，則用戶端裝置身分驗證不會收集指標。

預設：false

aggregatePeriodSeconds

以秒為單位的彙總期間，決定用戶端裝置身分驗證彙總指標並將其傳送至遙測代理程式的頻率。這不會變更指標發佈的頻率，因為遙測代理器仍會每天發佈一次。

預設：3600

startupTimeoutSeconds

(選用) 元件啟動的時間上限，以秒為單位。ERRORLED 如果元件超過此逾時，元件的狀態會變更為 。

預設：120

Example範例：組態合併更新 (使用限制性政策)

下列範例組態指定，允許名稱開頭為 的用戶端裝置MyClientDevice在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```

{
  "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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義用戶端裝置授權政策，以指定每個裝置群組的許可。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則至少包含一個選擇規則子句，這是單一表達式查詢，可以比對用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱結尾使用此萬用字元來比對名稱開頭為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

 Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \\。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名稱為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 `MyClientDevice` 的用戶端裝置。

```
thingName: MyClientDevice*
```

Example 範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

套用至此裝置群組中用戶端裝置的許可政策。指定您在 `policies` 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:`*deviceClientId* – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:`*mqttTopic* – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:`*mqttTopicFilter* – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 + 和 # MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 * 萬用字元，以允許存取所有動作。

`resources`

允許此政策中操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:`*mqttTopic*) 的清單。

您可以指定 * 萬用字元，以允許存取所有資源。您無法使用 * 萬用字元來比對部分資源識別符。例如，您可以指定 `"resources": "*"` ，但無法指定 `"resources": "mqtt:clientId:*"` 。

`statementDescription`

(選用) 此政策陳述式的描述。

`certificates`

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000 (10 天)

performance

(選用) 此核心裝置的效能組態選項。此物件包含下列資訊：

maxActiveAuthTokens

(選用) 作用中用戶端裝置授權字符的數量上限。您可以增加此數量，讓更多用戶端裝置連接到單一核心裝置，而無需重新驗證它們。

預設：2500

cloudRequestQueueSize

(選用) 此元件拒絕 AWS 雲端 請求之前佇列的請求數目上限。

預設：100

maxConcurrentCloudRequests

(選用) 要傳送至 的並行請求數目上限 AWS 雲端。您可以增加此數字，改善連接大量用戶端裝置的核心裝置上的身分驗證效能。

預設：1

certificateAuthority

(選用) 憑證授權機構組態選項，可將核心裝置中繼授權機構取代為您自己的中繼憑證授權機構。此物件包含下列資訊。

此物件包含下列資訊：

certificateUri

憑證的位置。它可以是檔案系統 URI 或指向存放在硬體安全模組中的憑證的 URI。

certificateChainUri

核心裝置 CA 的憑證鏈位置。這應該是返回根 CA 的完整憑證鏈。它可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證鏈。

privateKeyUri

核心裝置私有金鑰的位置。這可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證私有金鑰。

security

(選用) 此核心裝置的安全組態選項。此物件包含下列資訊。

clientDeviceTrustDurationMinutes

在使用核心裝置重新驗證之前，用戶端裝置的身分驗證資訊可信任的持續時間，以分鐘為單位。預設值為 1。

metrics

(選用) 此核心裝置的指標選項。只有在用戶端裝置驗證發生錯誤時，才會顯示錯誤指標。此物件包含下列資訊：

disableMetrics

如果 `disableMetrics` 欄位設定為 `true`，則用戶端裝置身分驗證不會收集指標。

預設：`false`

aggregatePeriodSeconds

以秒為單位的彙總期間，決定用戶端裝置身分驗證彙總指標並將其傳送至遙測代理程式的頻率。這不會變更指標發佈的頻率，因為遙測代理器仍會每天發佈一次。

預設：`3600`

Example範例：組態合併更新（使用限制性政策）

下列範例組態指定，允許名稱開頭為 `MyClientDevice` 的用戶端裝置在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```

{
  "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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義用戶端裝置授權政策，以指定每個裝置群組的許可。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則都包含至少一個選擇規則子句，這是單一表達式查詢，可以比對用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱結尾使用此萬用字元來比對名稱開頭為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\:\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 `MyClientDevice` 的用戶端裝置。

```
thingName: MyClientDevice*
```

Example 範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

policyName

套用至此裝置群組中用戶端裝置的許可政策。指定您在 `policies` 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:deviceClientId` – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:mqttTopic` – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:mqttTopicFilter` – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 `+` 和 `#` MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 `*` 萬用字元，以允許存取所有動作。

`resources`

允許此政策中操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:mqttTopic`) 的清單。

您可以指定 `*` 萬用字元，以允許存取所有資源。您無法使用 `*` 萬用字元來比對部分資源識別符。例如，您可以指定 `"resources": "*"` ，但無法指定 `"resources": "mqtt:clientId:*`。

`statementDescription`

(選用) 此政策陳述式的描述。

`certificates`

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000 (10 天)

performance

(選用) 此核心裝置的效能組態選項。此物件包含下列資訊：

maxActiveAuthTokens

(選用) 作用中用戶端裝置授權字符的數量上限。您可以增加此數量，讓更多用戶端裝置連接到單一核心裝置，而無需重新驗證它們。

預設：2500

cloudRequestQueueSize

(選用) 此元件拒絕 AWS 雲端 請求之前佇列的請求數目上限。

預設：100

maxConcurrentCloudRequests

(選用) 要傳送至 的並行請求數目上限 AWS 雲端。您可以增加此數字，改善連接大量用戶端裝置的核心裝置上的身分驗證效能。

預設：1

certificateAuthority

(選用) 憑證授權機構組態選項，可將核心裝置中繼授權機構取代為您自己的中繼憑證授權機構。此物件包含下列資訊。

certificateUri

憑證的位置。它可以是檔案系統 URI 或指向存放在硬體安全模組中的憑證的 URI。

certificateChainUri

核心裝置 CA 的憑證鏈位置。這應該是返回根 CA 的完整憑證鏈。它可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證鏈。

privateKeyUri

核心裝置私有金鑰的位置。這可以是檔案系統 URI 或 URI，指向存放在硬體安全模組中的憑證私有金鑰。

security

(選用) 此核心裝置的安全組態選項。此物件包含下列資訊。

clientDeviceTrustDurationMinutes

在需要使用核心裝置重新驗證之前，用戶端裝置的身分驗證資訊可信任的持續時間，以分鐘為單位。預設值為 1。

Example範例：組態合併更新（使用限制性政策）

下列範例組態指定，允許名稱開頭為 的用戶端裝置MyClientDevice在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```

{
  "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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義指定每個裝置群組許可的用戶端裝置授權政策。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則至少包含一個選擇規則子句，這是單一表達式查詢，可以比對用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱結尾使用此萬用字元來比對名稱開頭為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

 Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \\。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\:\\\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名稱為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 `MyClientDevice` 的用戶端裝置。

```
thingName: MyClientDevice*
```

Example 範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

套用至此裝置群組中用戶端裝置的許可政策。指定您在 `policies` 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:deviceClientId` – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:mqttTopic` – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:`*mqttTopicFilter* – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 `+` 和 `#` MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 `*` 萬用字元，以允許存取所有動作。

`resources`

允許此政策中操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:`*mqttTopic*) 的清單。

您可以指定 `*` 萬用字元，以允許存取所有資源。您無法使用 `*` 萬用字元來比對部分資源識別符。例如，您可以指定 `"resources": "*"` ，但無法指定 `"resources": "mqtt:clientId:*"`。

`statementDescription`

(選用) 此政策陳述式的描述。

`certificates`

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

`serverCertificateValiditySeconds`

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線並重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000 (10 天)

performance

(選用) 此核心裝置的效能組態選項。此物件包含下列資訊：

maxActiveAuthTokens

(選用) 作用中用戶端裝置授權字符的數量上限。您可以增加此數量，讓更多用戶端裝置連接到單一核心裝置，而無需重新驗證它們。

預設：2500

cloudRequestQueueSize

(選用) 此元件拒絕 AWS 雲端 請求之前佇列的請求數目上限。

預設：100

maxConcurrentCloudRequests

(選用) 要傳送至 的並行請求數目上限 AWS 雲端。您可以增加此數字，改善連接大量用戶端裝置的核心裝置上的身分驗證效能。

預設：1

Example範例：組態合併更新 (使用限制性政策)

下列範例組態指定，允許名稱開頭為 的用戶端裝置MyClientDevice在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```
{
  "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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義用戶端裝置授權政策，以指定每個裝置群組的許可。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則至少包含一個選擇規則子句，這是單一表達式查詢，可以符合用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱結尾使用此萬用字元來比對名稱開頭為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 (\) 逸出冒號 \。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\:\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example 範例選取規則

下列選擇規則符合名為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 `MyClientDevice` 的用戶端裝置。

```
thingName: MyClientDevice*
```

Example 範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

policyName

套用至此裝置群組中用戶端裝置的許可政策。指定您在 `policies` 物件中定義的政策名稱。

policies

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:deviceClientId` – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:mqttTopic` – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:mqttTopicFilter` – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 `+` 和 `#` MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 `*` 萬用字元，以允許存取所有動作。

resources

允許此政策中操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:mqttTopic`) 的清單。

您可以指定 `*` 萬用字元，以允許存取所有資源。您無法使用 `*` 萬用字元來比對部分資源識別符。例如，您可以指定 `"resources": "*"` ，但無法指定 `"resources": "mqtt:clientId:*`。

statementDescription

(選用) 此政策陳述式的描述。

certificates

(選用) 此核心裝置的憑證組態選項。此物件包含下列資訊：

serverCertificateValiditySeconds

(選用) 本機 MQTT 伺服器憑證過期的時間量 (以秒為單位)。您可以設定此選項來自訂用戶端裝置中斷連線和重新連線至核心裝置的頻率。

此元件會在本機 MQTT 伺服器憑證過期前 24 小時輪換本機 MQTT 伺服器憑證。MQTT 代理程式，例如 [Moquette MQTT 代理程式元件](#)，會產生新的憑證並重新啟動。發生這種情況時，所有連接到此核心裝置的用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至核心裝置。

預設：604800(7 天)

最小值：172800(2 天)

最大值：864000(10 天)

Example 範例：組態合併更新（使用限制性政策）

下列範例組態指定，允許名稱開頭為 MyClientDevice 的用戶端裝置在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```

{
  "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

裝置群組是具有與核心裝置連線和通訊許可的用戶端裝置群組。使用選取規則來識別用戶端裝置的群組，並定義指定每個裝置群組許可的用戶端裝置授權政策。

此物件包含下列資訊：

formatVersion

此組態物件的格式版本。

您可以從以下選項中選擇：

- 2021-03-05

definitions

此核心裝置的裝置群組。每個定義都會指定選擇規則，以評估用戶端裝置是否為群組的成員。每個定義也會指定要套用至符合選取規則之用戶端裝置的許可政策。如果用戶端裝置是多個裝置群組的成員，則裝置的許可會包含每個群組的許可政策。

此物件包含下列資訊：

groupNameKey

此裝置群組的名稱。將 *groupNameKey* 取代為可協助您識別此裝置群組的名稱。

此物件包含下列資訊：

selectionRule

指定哪些用戶端裝置是此裝置群組成員的查詢。當用戶端裝置連線時，核心裝置會評估此選擇規則，以判斷用戶端裝置是否為此裝置群組的成員。如果用戶端裝置是成員，核心裝置會使用此裝置群組的政策來授權用戶端裝置的動作。

每個選擇規則都包含至少一個選擇規則子句，這是單一表達式查詢，可以比對用戶端裝置。選擇規則使用與 AWS IoT 機群索引相同的查詢語法。如需選取規則語法的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 機群索引查詢語法](#)。

使用 * 萬用字元來比對具有一個選取規則子句的多個用戶端裝置。您可以在物件名稱結尾使用此萬用字元，來比對名稱開頭為您所指定字串的用戶端裝置。您也可以使用此萬用字元來比對所有用戶端裝置。

 Note

若要選取包含冒號字元 (:) 的值，請使用反斜線字元 () 逸出冒號\\。在 JSON 等格式中，您必須逸出反斜線字元，以便在冒號字元之前輸入兩個反斜線字元。例如，指定 `thingName: MyTeam\\\\\\:ClientDevice1` 選取名稱為 `MyTeam:ClientDevice1` 的物件。

您可以指定下列選擇器：

- `thingName` – 用戶端裝置 AWS IoT 物件的名稱。

Example範例選取規則

下列選擇規則符合名稱為 `MyClientDevice1` 或 `MyClientDevice2` 的用戶端裝置。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example範例選取規則（使用萬用字元）

下列選擇規則符合名稱開頭為 `MyClientDevice` 的用戶端裝置。

```
thingName: MyClientDevice*
```

Example範例選取規則（符合所有裝置）

下列選擇規則符合所有用戶端裝置。

```
thingName: *
```

`policyName`

套用至此裝置群組中用戶端裝置的許可政策。指定您在 `policies` 物件中定義的政策名稱。

`policies`

連接到核心裝置的用戶端裝置的用戶端裝置授權政策。每個授權政策都會指定一組動作，以及用戶端裝置可以執行這些動作的資源。

此物件包含下列資訊：

policyNameKey

此授權政策的名稱。將 *policyNameKey* 取代為可協助您識別此授權政策的名稱。您可以使用此政策名稱來定義適用於裝置群組的政策。

此物件包含下列資訊：

statementNameKey

此政策陳述式的名稱。將 *statementNameKey* 取代為可協助您識別此政策陳述式的名稱。

此物件包含下列資訊：

operations

允許此政策中資源的操作清單。

您可以包含下列任何操作：

- `mqtt:connect` – 准許連線至核心裝置。用戶端裝置必須具有此許可才能連線至核心裝置。

此操作支援下列資源：

- `mqtt:clientId:deviceClientId` – 根據用戶端裝置用來連線至核心裝置的 MQTT 代理程式的用戶端 ID 限制存取。將 *deviceClientId* 取代為要使用的用戶端 ID。
- `mqtt:publish` – 准許將 MQTT 訊息發佈至主題。

此操作支援下列資源：

- `mqtt:topic:mqttTopic` – 根據用戶端裝置發佈訊息的 MQTT 主題來限制存取。將 *mqttTopic* 取代為要使用的主題。

此資源不支援 MQTT 主題萬用字元。

- `mqtt:subscribe` – 准許訂閱 MQTT 主題篩選條件以接收訊息。

此操作支援下列資源：

- `mqtt:topicfilter:mqttTopicFilter` – 根據用戶端裝置可以訂閱訊息的 MQTT 主題來限制存取。將 *mqttTopicFilter* 取代為要使用的主題篩選條件。

此資源支援 + 和 # MQTT 主題萬用字元。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

用戶端裝置可以訂閱您允許的確切主題篩選條件。例如，如果您允許用戶端裝置訂閱 `mqtt:topicfilter:client/+/status` 資源，用戶端裝置可以訂閱 `client/+/status`，但不能訂閱 `client/client1/status`。

您可以指定 * 萬用字元，以允許存取所有動作。

resources

此政策中允許 操作的資源清單。指定與此政策中 操作對應的資源。例如，您可以在指定 `mqtt:publish` 操作的策略中指定 MQTT 主題資源 (`mqtt:topic:mqttTopic`) 的清單。

您可以指定 * 萬用字元，以允許存取所有資源。您無法使用 * 萬用字元來比對部分資源識別符。例如，您可以指定 **"resources": "*"，但無法指定 **"resources": "mqtt:clientId:*"****。

statementDescription

(選用) 此政策陳述式的描述。

Example 範例：組態合併更新 (使用限制性政策)

下列範例組態指定，允許名稱開頭為 `MyClientDevice` 的用戶端裝置在所有主題上連線和發佈/訂閱。

```
{
  "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範例：組態合併更新（使用寬鬆政策）

下列範例組態指定，以允許所有用戶端裝置在所有主題上連線和發佈/訂閱。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
```

```
"MyPermissivePolicy": {
  "AllowAll": {
    "statementDescription": "Allow client devices to perform all actions.",
    "operations": [
      "*"
    ],
    "resources": [
      "*"
    ]
  }
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.5.5	針對 Greengrass nucleus 2.16.0 版更新版本。
2.5.4	針對 Greengrass nucleus 2.15.0 版更新版本。
2.5.3	錯誤修正和改善 修正用戶端裝置因用戶端憑證過期而無法連線至核心裝置的問題。
2.5.2	針對 Greengrass nucleus 2.14.0 版更新版本。
2.5.1	錯誤修正和改善 支援 FIPS 端點。
2.5.0	新功能 - 允許政策資源的 <code>\${iot:Connection.Thing.ThingName}</code> 變數替換。 - 允許具有萬用字元的政策資源，例如 <code>mqtt:topic:my*</code>。
2.4.5	新功能 新增對萬用字元字首的支援，以使用 <code>selectionRule</code> 參數選取物件名稱。 錯誤修正和改善 修正在特定情況下，憑證未更新為新連線資訊的問題。
2.4.4	針對 Greengrass nucleus 2.12.0 版更新版本。
2.4.3	針對 Greengrass nucleus 2.11.0 版更新版本。
2.4.2	新功能 新增 <code>startupTimeoutSeconds</code> 組態選項。
2.4.1	針對 Greengrass nucleus 2.10.0 版更新版本。

版本	變更
2.4.0	新功能 - 新增對用戶端裝置身分驗證的支援，以發出將由遙測代理程式發佈的操作指標。 錯誤修正和改善 - 修正用戶端裝置身分驗證需要超過 10 秒才能驗證用戶端裝置身分的問題。 - 其他次要修正和改進。
2.3.2	錯誤修正和改善 新增快取主機名稱資訊的支援，讓元件在離線重新啟動時正確產生憑證主體。
2.3.1	錯誤修正和改善 修正記憶體洩漏。
2.3.0	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 新功能 - 新增用戶端裝置的離線身分驗證支援，以便在核心裝置未連線至網際網路時，繼續連線至核心裝置。 - 新增支援核心裝置用來產生 MQTT 代理程式憑證的客戶提供憑證授權單位做為根憑證。
2.2.3	針對 Greengrass nucleus 2.8.0 版更新版本。
2.2.2	錯誤修正和改善 修正本機 MQTT 伺服器憑證在特定情況下旋轉頻率高於預期的問題。
2.2.1	針對 Greengrass nucleus 2.7.0 版更新版本。

版本	變更
2.2.0	新功能 - 新增自訂元件的支援，以呼叫程序間通訊 (IPC) 操作來驗證和授權用戶端裝置。例如，您可以在自訂 MQTT 代理程式元件中使用這些操作。如需詳細資訊，請參閱 IPC：驗證和授權用戶端裝置。 - 新增您可以設定來調整此元件執行方式的 <code>maxActiveAuthTokens</code>、<code>cloudQueueSize</code>、和 <code>threadPoolSize</code> 選項。
2.1.0	新功能 - 新增您可以設定在 MQTT 代理程式伺服器憑證過期時自訂 <code>serverCertificateValiditySeconds</code> 的選項。您可以將伺服器憑證設定為在 2 到 10 天後過期。 錯誤修正和改善 - 修正此元件如何處理組態重設更新的問題。 - 修正本機 MQTT 伺服器憑證在特定情況下旋轉頻率高於預期的問題。 若要套用此修正，您還必須使用 Moquette MQTT 代理程式元件 的 v2.1.0 或更新版本。 - 改善此元件在輪換憑證時記錄的訊息。 - 針對 Greengrass nucleus 2.6.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.3	錯誤修正和改善 - 如果您輪換核心裝置的私有金鑰，登入資料現在會重新整理。 - 更新以讓日誌訊息更清楚。
2.0.2	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.1	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.0	初始版本。

CloudWatch 指標

Amazon CloudWatch 指標元件 (aws.greengrass.Cloudwatch) 會將自訂指標從 Greengrass 核心裝置發佈至 Amazon CloudWatch。元件可讓元件發佈 CloudWatch 指標，您可以用來監控和分析 Greengrass 核心裝置的環境。如需詳細資訊，請參閱《[Amazon CloudWatch 使用者指南](#)》中的[使用 Amazon CloudWatch 指標](#)。Amazon CloudWatch

若要使用此元件發佈 CloudWatch 指標，請將訊息發佈至此元件訂閱的主題。根據預設，此元件會訂閱 cloudwatch/metric/put [本機發佈/訂閱](#) 主題。您可以在部署此元件時指定其他主題，包括 AWS IoT Core MQTT 主題。

此元件會批次處理位於相同命名空間中的指標，並定期將其發佈至 CloudWatch。

Note

此元件提供與 AWS IoT Greengrass V1 中的 CloudWatch 指標連接器類似的功能。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的 [CloudWatch 指標連接器](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [輸出資料](#)
- [授權](#)
- [本機日誌檔案](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 3.2.x
- 3.1.x 版本
- 3.0.x
- 2.1.x
- 2.0.x

如需有關每個元件版本變更的資訊，請參閱[變更日誌](#)。

Type

v3.x

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

v2.x

此元件是 Lambda 元件 (aws.greengrass.lambda)。 [Greengrass 核](#) 會使用 Lambda [啟動器元件執行此元件的 Lambda 函數](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

v3.x

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

v2.x

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

3.x

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- [Greengrass 裝置角色](#) 必須允許 `cloudwatch:PutMetricData` 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

如需詳細資訊，請參閱《[Amazon CloudWatch 使用者指南](#)》中的 [Amazon CloudWatch 許可參考](#)。Amazon CloudWatch

2.x

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- [Greengrass 裝置角色](#) 必須允許 `cloudwatch:PutMetricData` 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

```

    ],
    "Effect": "Allow",
    "Resource": "*"
  }
]
}

```

如需詳細資訊，請參閱《[Amazon CloudWatch 使用者指南](#)》中的 [Amazon CloudWatch 許可參考](#)。Amazon CloudWatch

- 若要從此元件接收輸出資料，您必須在部署此元件時合併[舊版訂閱路由器元件](#) (aws.greengrass.LegacySubscriptionRouter) 的下列組態更新。此組態會指定此元件發佈回應的主題。

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"
    }
  }
}

```

- 以您使用 AWS 區域 的 取代##。

- 以此元件執行的 Lambda 函數版本取代 版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的配方。在[AWS IoT Greengrass 主控台](#)中開啟此元件的詳細資訊頁面，並尋找 Lambda 函數鍵值對。此鍵/值對包含 Lambda 函數的名稱和版本。

Important

每次部署此元件時，都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您為部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱[建立部署](#)。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
monitoring. <i>region</i> .amazonaws.com	443	是	上傳 CloudWatch 指標。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以在 [AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

3.2.0

下表列出此元件 3.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	>=0.0.0	硬式

3.0.0 - 3.1.0

下表列出此元件 3.0.0 到 3.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式
字符交換服務	>=0.0.0	硬式

2.1.4 - 2.1.9

下表列出此元件 2.1.4 至 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.4 - 2.1.8

下表列出此元件 2.1.4 和 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	硬式
Lambda 啟動器	^2.0.0	硬式

相依性	相容版本	相依性類型
Lambda 執行階段	^2.0.0	軟式
字交換服務	^2.0.0	硬式

2.1.2 - 2.1.3

下表列出此元件 2.1.2 和 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.8 - 2.1.0

下表列出此元件 2.0.8 至 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	硬式
Lambda 啟動器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	>=1.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v3.x

PublishInterval

(選用) 在元件發佈指定命名空間的批次指標之前等待的秒數上限。若要設定元件在接收指標時發佈指標，這表示無需批次處理，請指定 0。

在相同命名空間中收到 20 個指標或您指定的間隔之後，元件會發佈到 CloudWatch。

Note

元件不會指定事件發佈的順序。

此值最多可達 900 秒。

預設：10 秒

MaxMetricsToRetain

(選用) 在元件將指標取代為較新的指標之前，所有命名空間中要儲存在記憶體中的指標數量上限。

當核心裝置沒有網際網路連線時，就會套用此限制，因此元件會緩衝指標以供稍後發佈。當緩衝區已滿時，元件會將最舊的指標取代為較新的指標。指定命名空間中的指標只會取代相同命名空間中的指標。

Note

如果元件的主機程序中斷，則元件不會儲存指標。例如，這可能會在部署期間或核心裝置重新啟動時發生。

此值必須至少為 2,000 個指標。

預設：5,000 個指標

InputTopic

(選用) 元件訂閱接收訊息的主題。如果您 `true` 為指定 `PubSubToIoTCore`，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

預設：cloudwatch/metric/put

OutputTopic

(選用) 元件發佈狀態回應的主題。

預設：cloudwatch/metric/put/status

PubSubToIoTCore

(選用) 字串值，定義是否發佈和訂閱 AWS IoT Core MQTT 主題。支援的值為 `true` 和 `false`。

預設：false

LogLevel

(選用) 元件的記錄層級。從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARNING
- ERROR
- CRITICAL

預設：INFO

UseInstaller

(選用) 布林值，定義是否使用此元件中的安裝程式指令碼來安裝此元件的 SDK 相依性。

`false` 如果您想要使用自訂指令碼安裝相依性，或想要在預先建置的 Linux 映像中包含執行時間相依性，請將此值設定為 `true`。若要使用此元件，您必須安裝下列程式庫，包括任何相依性，並將其提供給預設 Greengrass 系統使用者。

- [AWS IoT Device SDK 適用於 Python 的 v2](#)
- [適用於 Python \(Boto3\) 的 AWS SDK](#)

預設：true

PublishRegion

(選用) 要發佈 CloudWatch 指標 AWS 區域的。此值會覆寫核心裝置的預設區域。只有跨區域指標才需要此參數。

accessControl

(選用) 包含[授權政策](#)的物件，允許元件發佈和訂閱指定的主題。如果您為 InputTopic 和指定自訂值 OutputTopic，則必須更新此物件中的資源值。

預設：

```
{
  "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範例：組態合併更新

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

Note

此元件的預設組態包含 Lambda 函數參數。建議您只編輯下列參數，以在裝置上設定此元件。

lambdaParams

包含此元件 Lambda 函數參數的物件。此物件包含下列資訊：

EnvironmentVariables

包含 Lambda 函數參數的物件。此物件包含下列資訊：

PUBLISH_INTERVAL

(選用) 在元件發佈指定命名空間的批次指標之前等待的秒數上限。若要設定元件在接收指標時發佈指標，這表示無需批次處理，請指定 0。

在相同命名空間中收到 20 個指標或您指定的間隔之後，元件會發佈到 CloudWatch。

 Note

元件不保證事件發佈的順序。

此值最多可達 900 秒。

預設：10 秒

MAX_METRICS_TO_RETAIN

(選用) 元件將指標取代為較新的指標之前，所有命名空間中要儲存在記憶體中的指標數量上限。

此限制適用於核心裝置沒有網際網路連線的情況，因此元件會緩衝稍後發佈的指標。當緩衝區已滿時，元件會將最舊的指標取代為較新的指標。指定命名空間中的指標只會取代相同命名空間中的指標。

 Note

如果元件的主機程序中斷，則元件不會儲存指標。例如，這可能發生在部署期間或核心裝置重新啟動時。

此值必須至少為 2,000 個指標。

預設：5,000 個指標

PUBLISH_REGION

(選用) 要發佈 CloudWatch 指標 AWS 區域的。此值會覆寫核心裝置的預設區域。只有跨區域指標需要此參數。

containerMode

(選用) 此元件的容器化模式。您可以從以下選項中選擇：

- NoContainer – 元件不會在隔離的執行時間環境中執行。
- GreengrassContainer – 元件會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

預設：GreengrassContainer

containerParams

(選用) 包含此元件容器參數的物件。如果您 `GreengrassContainer` 為指定，元件會使用這些參數 `containerMode`。

此物件包含下列資訊：

`memorySize`

(選用) 要配置給元件的記憶體量 (以 KB 為單位)。

預設為 64 MB (65,535 KB)。

pubsubTopics

(選用) 物件，其中包含元件訂閱接收訊息的主題。您可以指定每個主題，以及元件是從訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

`type`

(選用) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- `PUB_SUB` - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，主題不能包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- `IOT_CORE` - 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設：`PUB_SUB`

`topic`

(選用) 元件訂閱接收訊息的主題。如果您 `IotCore` 為指定 `type`，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

Example 範例：組態合併更新 (容器模式)

```
{
```

```
"containerMode": "GreengrassContainer"
}
```

Example範例：組態合併更新（無容器模式）

```
{
  "containerMode": "NoContainer"
}
```

輸入資料

此元件接受下列主題的指標，並將指標發佈至 CloudWatch。根據預設，此元件會訂閱本機發佈/訂閱訊息。如需如何從自訂元件發佈訊息至此元件的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。

從元件版本 v3.0.0 開始，您可以選擇將此元件設定為訂閱 MQTT 主題，方法是將 PubSubToIoTCore 組態參數設定為 true。如需將訊息發佈至自訂元件中 MQTT 主題的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設主題：cloudwatch/metric/put

訊息接受下列屬性。輸入訊息必須是 JSON 格式。

request

此訊息中的指標。

請求物件包含要發佈至 CloudWatch 的指標資料。指標值必須符合 [PutMetricData](#) 操作的規格。

類型：object 包含下列資訊：

namespace

此請求中指標資料的使用者定義命名空間。CloudWatch 使用命名空間做為指標資料點的容器。

Note

您無法指定以預留字串 開頭的命名空間 AWS/。

類型：string

有效模式：`[^:].*`

metricData

指標的資料。

類型：object 包含下列資訊：

metricName

指標的名稱

類型：string

value

指標的值。

Note

CloudWatch 會拒絕太小或太大的值。值必須介於 $8.515920e-109$ 和 $1.174271e+108$ (Base 10) 或 $2e-360$ 和 $2e360$ (Base 2) 之間。CloudWatch 不支援特殊值，例如 NaN、+Infinity 和 -Infinity。

類型：double

dimensions

(選用) 指標的維度。維度提供指標及其資料的其他資訊。一個指標可以定義最多 10 個維度。

此元件會自動包含名為 `coreName` 的維度，其中 `coreName` 值是核心裝置的名稱。

類型：array 的物件，每個物件都包含下列資訊：

name

(選用) 維度名稱。

類型：string

value

(選用) 維度值。

類型：string

timestamp

(選用) 收到指標資料的時間，以秒為單位，以 Unix epoch 時間表示。

預設為元件接收訊息的時間。

類型：double

Note

如果您在此元件的 2.0.3 和 2.0.7 版之間使用，我們建議您從單一來源傳送多個指標時，分別擷取每個指標的時間戳記。請勿使用變數來存放時間戳記。

unit

(選用) 指標的單位。

類型：string

有效

值：Seconds、Microseconds、Milliseconds、Bytes、Kilobytes、MegabytesGigab
Second、Kilobytes/Second、Megabytes/Second、Gigabytes/
SecondTerabytes/SecondBits/SecondKilobits/Second、Megabits/Second
Gigabits/Second Terabits/Second Count/Second None

預設為 None。

Note

套用至 CloudWatch PutMetricData API 的所有配額都會套用至您使用此元件發佈的指標。
下列配額特別重要：

- API 承載的 40 KB 限制
- 每一 API 請求 20 個指標
- PutMetricData API 每秒 150 筆交易 (TPS)

如需詳細資訊，請參閱 [《CloudWatch 使用者指南》中的 CloudWatch 服務配額](#)。CloudWatch

Example範例輸入

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
  }
}
```

輸出資料

根據預設，此元件會將回應發佈為下列本機發佈/訂閱主題上的輸出資料。如需如何在自訂元件中訂閱此主題訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。

您可以選擇將此元件設定為發佈至 MQTT 主題，方法是將PubSubToIoTCore組態參數設定為 true。如需在自訂元件中訂閱 MQTT 主題訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

Note

根據預設，元件版本 2.0.x 會將回應發佈為 MQTT 主題上的輸出資料。您必須在[舊版訂閱路由](#)器元件的組態subject中，將主題指定為。

預設主題：cloudwatch/metric/put/status

Example範例輸出：成功

回應包含指標資料的命名空間，以及 CloudWatch 回應中的 RequestId 欄位。

```
{
```

```
"response": {
  "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
  "namespace": "Greengrass",
  "status": "success"
}
```

Example範例輸出：失敗

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

Note

如果元件偵測到可以重試的錯誤，例如連線錯誤，則會在下一個批次中重試發佈。

授權

此元件包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C:\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

v3.x

版本	變更
3.2.0	新功能 新增 Greengrass nucleus lite 的配方支援

版本	變更
3.1.0	錯誤修正和改善 新增對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理及讓核心裝置信任 HTTPS 代理。
3.0.0	此版本的 CloudWatch 指標元件預期與 2.x 版不同的組態參數。如果您使用 2.x 版的非預設組態，而且想要從 v2.x 升級到 v3.x，則必須更新元件的組態。如需詳細資訊，請參閱 CloudWatch 指標元件組態。 新功能 - 新增對執行 Windows 之核心裝置的支援。 - 將元件類型從 Lambda 元件變更為一般元件。此元件現在不再依賴舊版訂閱路由器元件來建立訂閱。 - 新增InputTopic 組態參數，以指定元件訂閱接收訊息的主題。 - 新增OutputTopic 組態參數，以指定元件發佈狀態回應的主題。 - 新增PubSubToIoTCore 組態參數，以指定是否發佈和訂閱 AWS IoT Core MQTT 主題。 - 新增新的UseInstaller 組態參數，可讓您選擇性地停用安裝元件相依性的安裝指令碼。 錯誤修正和改善 新增對輸入資料中重複時間戳記的支援。

v2.x

版本	變更
2.1.8	針對 Greengrass nucleus 2.13.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.6.0 版更新版本。

版本	變更
2.1.0	新功能 • 新增對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理及讓核心裝置信任 HTTPS 代理。
2.0.8	錯誤修正和改善 • 新增對輸入資料中重複時間戳記的支援。 • 針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

另請參閱

- 《Amazon CloudWatch 使用者指南》中的[使用 Amazon CloudWatch 指標](#)
- Amazon CloudWatch API 參考中的[PutMetricData](#)

AWS IoT Device Defender

AWS IoT Device Defender 元件 (`aws.greengrass.DeviceDefender`) 會通知管理員有關 Greengrass 核心裝置狀態的變更。這可協助識別可能會表示裝置受損的不尋常行為。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT Device Defender](#)。

此元件會讀取核心裝置上的系統指標。然後，它會將指標發佈至 AWS IoT Device Defender。如需如何讀取和解譯此元件所報告指標的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[裝置指標文件規格](#)。

 Note

此元件提供與 Device Defender 連接器類似的功能 AWS IoT Greengrass V1。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的 [Device Defender 連接器](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [輸出資料](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 3.1.x 版本
- 3.0.x
- 2.0.x

如需有關每個元件版本變更的資訊，請參閱[變更日誌](#)。

類型

v3.x

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

v2.x

此元件是 Lambda 元件 (`aws.greengrass.lambda`)。 [Greengrass 核](#)會使用 Lambda [啟動器元件執行此元件的 Lambda 函數](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

v3.x

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

v2.x

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

v3.x

- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- AWS IoT Device Defender 設定為使用 Detect 功能來監控違規。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[偵測](#)。

v2.x

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- AWS IoT Device Defender 設定為使用 Detect 功能來監控違規。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[偵測](#)。

- 安裝在核心裝置上的 [psutil](#) 程式庫。5.7.0 版是經過驗證可與 元件搭配使用的最新版本。
- 安裝在核心裝置上的 [cbor](#) 程式庫。1.0.0 版是經過驗證可與 元件搭配使用的最新版本。
- 若要從此元件接收輸出資料，您必須在部署此元件時合併[舊版訂閱路由器元件](#) (aws.greengrass.LegacySubscriptionRouter) 的下列組態更新。此組態會指定此元件發佈回應的主題。

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"
    }
  }
}
```

- 以您使用 AWS 區域 的 取代##。
- 以此元件執行的 Lambda 函數版本取代 版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的配方。在[AWS IoT Greengrass 主控台](#)中開啟此元件的詳細資訊頁面，並尋找 Lambda 函數鍵/值對。此鍵/值對包含 Lambda 函數的名稱和版本。

⚠ Important

每次部署此元件時，都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保您為部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱[建立部署](#)。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以在 [AWS IoT Greengrass 主控台](#) 中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

3.1.1

下表列出此元件 3.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式
字符交換服務	>=0.0.0	硬式

3.0.0 - 3.0.2

下表列出此元件 3.0.0 到 3.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式
權杖交換服務	>=0.0.0	硬式

2.0.12 - 2.0.17

下表列出此元件 2.0.12 至 2.0.17 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.12 - 2.0.16

下表列出此元件 2.0.16 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.10 - 2.0.11

下表列出此元件 2.0.10 和 2.0.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	相容版本	相依性類型
字符交換服務	^2.0.0	硬式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字元交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	硬式
Lambda 啟動器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式
權杖交換服務	>=1.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v3.x

PublishRetryCount

將重試發佈的次數。此功能可在 3.1.1 版中使用。

最小值為 0。

上限為 72。

預設：5

`SampleIntervalSeconds`

(選用) 元件收集和報告指標的每個週期之間的秒數。

最低值為 300 秒 (5 分鐘)。

預設：300 秒

`UseInstaller`

(選用) 布林值，定義是否使用此元件中的安裝程式指令碼來安裝此元件的相依性。

`false` 如果您想要使用自訂指令碼來安裝相依性，或想要在預先建置的 Linux 映像中包含執行時間相依性，請將此值設定為 `false`。若要使用此元件，您必須安裝下列程式庫，包括任何相依性，並將其提供給預設的 Greengrass 系統使用者。

- [AWS IoT Device SDK 適用於 Python 的 v2](#)
- [cbor](#) 程式庫。1.0.0 版是經過驗證可與 元件搭配使用的最新版本。
- [psutil](#) 程式庫。5.7.0 版是經過驗證可與 元件搭配使用的最新版本。

 Note

如果您在設定為使用 HTTPS 代理的核心裝置上使用此元件的 3.0.0 或 3.0.1 版，您必須將此值設定為 `false`。在此元件的這些版本中，安裝程式指令碼不支援 HTTPS 代理背後的操作。

預設：`true`

v2.x

 Note

此元件的預設組態包含 Lambda 函數參數。建議您只編輯下列參數，以在裝置上設定此元件。

lambdaParams

包含此元件 Lambda 函數參數的物件。此物件包含下列資訊：

EnvironmentVariables

包含 Lambda 函數參數的物件。此物件包含下列資訊：

PROCFS_PATH

(選用) /proc 資料夾的路徑。

- 若要在容器中執行此元件，請使用預設值 /host-proc。根據預設，元件會在容器中執行。
- 若要在沒有容器模式下執行此元件，請 /proc 為此參數指定。

預設：/host-proc。這是此元件在容器中掛載 /proc 資料夾的預設路徑。

Note

此元件具有此資料夾的唯讀存取權。

SAMPLE_INTERVAL_SECONDS

(選用) 元件收集和報告指標的每個週期之間的秒數。

最低值為 300 秒 (5 分鐘)。

預設：300 秒

containerMode

(選用) 此元件的容器化模式。您可以從以下選項中選擇：

- GreengrassContainer – 元件會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。
- NoContainer – 元件不會在隔離的執行時間環境中執行。

如果您指定此選項，則必須 /proc 為 PROCFS_PATH 環境變數參數指定。

預設：GreengrassContainer

containerParams

(選用) 包含此元件容器參數的物件。如果您GreengrassContainer為指定，元件會使用這些參數containerMode。

此物件包含下列資訊：

memorySize

(選用) 要配置給元件的記憶體量 (以 KB 為單位)。

預設為 50,000 KB。

pubsubTopics

(選用) 物件，其中包含元件訂閱接收訊息的主題。您可以指定每個主題，以及元件是從或 AWS IoT Core 本機發佈/訂閱主題訂閱 MQTT 主題。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

type

(選用) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，主題不能包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- IOT_CORE – 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設：PUB_SUB

topic

(選用) 元件訂閱接收訊息的主題。如果您IotCore為指定 type，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

Example範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
```



```
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    },
    "containerMode": "GreengrassContainer"
  }
```

Example範例：組態合併更新（無容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

輸入資料

此元件不接受訊息做為輸入資料。

輸出資料

此元件會將安全指標發佈至 的下列預留主題 AWS IoT Device Defender。此元件會在發佈指標時，以核心裝置的名稱取代 *coreDeviceName*。

主題 (AWS IoT Core MQTT)：\$aws/things/*coreDeviceName*/defender/metrics/json

Example範例輸出

```
{
  "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  
  }  
}  
}
```

如需此元件報告之指標的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[裝置指標文件規格](#)。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

授權

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

v3.x

版本	變更
3.1.1	錯誤修正和改善 - 當連線在網路中斷後無法復原時，新增用戶端連線的重試次數。 - 新增可設定的重試以發佈指標。
3.1.0	錯誤修正和改善 新增對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理及讓核心裝置信任 HTTPS 代理。
3.0.1	修正元件如何計算指標差異值的問題。
3.0.0	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 初始版本。

v2.x

版本	變更
2.0.17	針對 Greengrass nucleus 2.14.0 版更新版本。
2.0.16	針對 Greengrass nucleus 2.13.0 版更新版本。
2.0.11	針對 Greengrass nucleus 2.11.0 版更新版本。
2.0.10	針對 Greengrass nucleus 2.7.0 版更新版本。
2.0.9	針對 Greengrass nucleus 2.6.0 版更新版本。

版本	變更
2.0.8	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

磁碟多工緩衝處理

磁碟多工緩衝處理元件 (`aws.greengrass.DiskSpooler`) 為從 Greengrass 核心裝置多工緩衝處理的訊息提供持久性儲存選項 AWS IoT Core。此元件會將這些傳出訊息存放在磁碟上。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [用途](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 1.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核會在](#)與核相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- storageType 應設定為 Disk 以使用此元件。您可以在 [Greengrass 核組態](#) 中設定此項目。
- maxSizeInBytes 不得設定為大於裝置上的可用空間。您可以在 [Greengrass 核組態](#) 中設定此項目。
- 磁碟多工緩衝處理常式元件支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

1.0.7

下表列出此元件 1.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.17.0	硬式

1.0.

下表列出此元件 1.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.16.0	硬式

1.0.5

下表列出此元件 1.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.15.0	硬式

1.0.4

下表列出此元件 1.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.14.0	硬式

1.0.1 – 1.0.3

下表列出此元件 1.0.1 至 1.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.13.0	硬式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.12.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

用途

若要使用磁碟多工緩衝處理元件，`aws.greengrass.DiskSpooler`必須部署。

若要設定和使用此元件，您必須將 `pluginName` 設定為 `aws.greengrass.DiskSpooler`。

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```


Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.0.7	針對 Greengrass nucleus 2.16.0 版更新版本。
1.0.6	針對 Greengrass nucleus 2.15.0 版更新版本。
1.0.5	針對 Greengrass nucleus 2.14.0 版更新版本。
1.0.4	錯誤修正和改善 一般錯誤修正。
1.0.3	錯誤修正和改善 透過重複使用資料庫連線來改善效能。
1.0.2	錯誤修正和改善 修正在某些情況下未保留 MQTT 訊息格式欄位的問題。
1.0.1	針對 Greengrass nucleus 2.12.0 版更新版本。
1.0.0	初始版本。

Docker 應用程式管理員

Docker 應用程式管理員元件 (aws.greengrass.DockerApplicationManager) 可讓從 Amazon Elastic Container Registry (Amazon ECR) 上託管的公 AWS IoT Greengrass 有映像登錄檔和私有登錄檔下載 Docker 映像。它還允許自動 AWS IoT Greengrass 管理登入資料，以安全地從 Amazon ECR 中的私有儲存庫下載映像。

當您開發執行 Docker 容器的自訂元件時，請包含 Docker 應用程式管理員做為下載元件中指定為成品之 Docker 映像的相依性。如需詳細資訊，請參閱[執行 Docker 容器](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- [Docker 引擎](#) 1.9.1 或更新版本安裝在 Greengrass 核心裝置上。20.10 版是經過驗證可與 AWS IoT Greengrass Core 軟體搭配使用的最新版本。您必須先直接在核心裝置上安裝 Docker，才能部署執行 Docker 容器的元件。
- 在部署此元件之前，Docker 協助程式會在核心裝置上啟動和執行。
- 存放在下列其中一個支援映像來源中的 Docker 映像：
 - Amazon Elastic Container Registry (Amazon ECR) 中的公有和私有映像儲存庫
 - 公有 Docker Hub 儲存庫
 - 公有 Docker 信任的登錄檔
- Docker 映像包含在自訂 Docker 容器元件中做為成品。使用下列 URI 格式來指定 Docker 映像：
 - 私有 Amazon ECR 映像：`docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag@digest]`
 - 公有 Amazon ECR 映像：`docker:public.ecr.aws/repository/image[:tag@digest]`
 - 公有 Docker Hub 映像：`docker:name[:tag@digest]`

如需詳細資訊，請參閱[執行 Docker 容器](#)。

Note

如果您未在映像的成品 URI 中指定映像標籤或映像摘要，則當您部署自訂 Docker 容器元件時，Docker 應用程式管理員會提取該映像的最新可用版本。為了確保您的所有核心裝置執行相同版本的映像，建議您在成品 URI 中包含映像標籤或映像摘要。

- 執行 Docker 容器元件的系統使用者必須具有根或管理員許可，或者您必須將 Docker 設定為以非根或非註冊使用者身分執行。
 - 在 Linux 裝置上，您可以將使用者新增至 `docker` 群組，在沒有 `sudo` 的情況下呼叫 `docker` 命令。
 - 在 Windows 裝置上，您可以將使用者新增至 `docker-users` 群組，以呼叫沒有管理員權限的 `docker` 命令。

Linux or Unix

若要將 `ggc_user` 或您用來執行 Docker 容器元件的非根使用者新增至 `docker` 群組，請執行下列命令。

```
sudo usermod -aG docker ggc_user
```

如需詳細資訊，請參閱[以非根使用者的身分管理 Docker](#)。

Windows Command Prompt (CMD)

若要將 `ggc_user` 或您用來執行 Docker 容器元件的使用者新增至 `docker-users` 群組，請以管理員身分執行下列命令。

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

若要將 `ggc_user` 或您用來執行 Docker 容器元件的使用者新增至 `docker-users` 群組，請以管理員身分執行下列命令。

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- 如果您將 [AWS IoT Greengrass Core 軟體設定為使用網路代理](#)，則必須將 [Docker 設定為使用相同的代理伺服器](#)。
- 如果您的 Docker 映像存放在 Amazon ECR 私有登錄檔中，則您必須在 Docker 容器元件中包含字符交換服務元件做為相依性。此外，[Greengrass 裝置角色](#) 必須允許 `ecr:GetAuthorizationToken`、`ecr:BatchGetImage` 和 `ecr:GetDownloadUrlForLayer` 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

```
}
```

- 支援在 VPC 中執行 docker 應用程式管理員元件。若要在 VPC 中部署此元件，需要下列項目。
 - Docker 應用程式管理員元件必須具有下載映像的連線能力。例如，如果您使用 ECR，您必須連線至下列端點。
 - *.dkr.ecr.**region**.amazonaws.com (VPC 端點 com.amazonaws.**region**.ecr.dkr)
 - api.ecr.**region**.amazonaws.com (VPC 端點 com.amazonaws.**region**.ecr.api)

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
ecr. region .amazonaws.com	443	否	如果您從 Amazon ECR 下載 Docker 映像，則為必要項目。
hub.docker.com registry.hub.docker.com/v1	443	否	如果您從 Docker Hub 下載 Docker 映像，則為必要項目。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.0.15

下表列出此元件 2.0.15 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.17.0	軟式

2.0.14

下表列出此元件 2.0.14 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.16.0	軟式

2.0.13

下表列出此元件 2.0.13 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.15.0	軟式

2.0.12

下表列出此元件 2.0.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.14.0	軟式

2.0.11

下表列出此元件 2.0.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.13.0	軟式

2.0.10

下表列出此元件 2.0.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.12.0	軟式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.11.0	軟式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.10.0	軟式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.9.0	軟式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.8.0	軟式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.7.0	軟式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.6.0	軟式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.5.0	軟式

2.0.2

下表列出此元件 2.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.4.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.3.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.2.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.0.15	針對 Greengrass nucleus 2.16.0 版更新版本。
2.0.14	針對 Greengrass nucleus 2.15.0 版更新版本。
2.0.13	針對 Greengrass nucleus 2.14.0 版更新版本。
2.0.12	針對 Greengrass nucleus 2.13.0 版更新版本。
2.0.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.0.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.0.9	針對 Greengrass nucleus 2.10.0 版更新版本。
2.0.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.7.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.6.0 版更新版本。

版本	變更
2.0.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.0	初始版本。

另請參閱

- [執行 Docker 容器](#)

Kinesis Video Streams 的邊緣連接器

Kinesis Video Streams 元件 (`aws.iot.EdgeConnectorForKVS`) 的邊緣連接器會從本機攝影機讀取影片饋送，並將串流發佈至 Kinesis Video Streams。您可以將此元件設定為使用即時串流通訊協定 (RTSP) 從網際網路通訊協定 (IP) 攝影機讀取視訊摘要。然後，您可以在 [Amazon Managed Grafana](#) 或本機 Grafana 伺服器中設定儀表板，以監控視訊串流並與之互動。

您可以將此元件與 AWS IoT TwinMaker 整合，以在 Grafana 儀表板中顯示和控制影片串流。AWS IoT TwinMaker 是一項 AWS 服務，可讓您建置實體系統的操作數位分身。您可以使用 AWS IoT TwinMaker 從感應器、攝影機和企業應用程式視覺化資料，以追蹤實體工廠、建築物或工業工廠。您也可以使用此資料來監控操作、診斷錯誤和修復錯誤。如需詳細資訊，請參閱 [AWS IoT TwinMaker 使用者指南中的什麼是TwinMaker？](#)。AWS IoT TwinMaker

此元件會將其組態存放在 `中` AWS IoT SiteWise，這是建立模型並存放工業資料 AWS 的服務。在 `中` AWS IoT SiteWise，資產代表物件，例如裝置、設備或其他物件的群組。若要設定和使用此元件，您可以為每個 Greengrass 核心裝置和連接到每個核心裝置的每個 IP 攝影機建立 AWS IoT SiteWise 資產。每個資產都有您設定的屬性來控制功能，例如即時串流、隨需上傳和本機快取。若要指定每個攝影機的 URL，您可以在 AWS Secrets Manager 中建立包含攝影機 URL 的秘密。如果攝影機需要身分驗證，您也可以指定使用者名稱和密碼。然後，您可以在 IP 攝影機的資產屬性中指定該秘密。

此元件會將每個相機的影片串流上傳至 Kinesis 影片串流。您可以在每個攝影機的 AWS IoT SiteWise 資產組態中指定目的地 Kinesis 視訊串流的名稱。如果 Kinesis 影片串流不存在，此元件會為您建立。

AWS IoT TwinMaker 提供指令碼，您可以執行此指令碼來建立這些 AWS IoT SiteWise 資產和 Secrets Manager 秘密。如需如何建立這些資源以及如何安裝、設定和使用此元件的詳細資訊，請參閱 [AWS IoT TwinMaker 使用者指南中的 TwinMaker 影片整合](#)。AWS IoT TwinMaker

Note

Kinesis Video Streams 元件的邊緣連接器僅適用於下列 AWS 區域：

- 美國東部 (維吉尼亞北部)
- 美國西部 (奧勒岡)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 亞太地區 (新加坡)
- 亞太地區 (東京)
- 亞太地區 (首爾)
- 亞太地區 (悉尼)
- 亞太地區 (孟買)
- 中國 (北京)

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [授權](#)
- [Usage](#)
- [本機日誌檔案](#)
- [變更記錄](#)

- [另請參閱](#)

版本

此元件具有下列版本：

- 1.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 您只能將此元件部署至單一核心裝置，因為每個核心裝置的元件組態必須是唯一的。您無法將此元件部署至核心裝置的群組。
- 核心裝置上已安裝 [GStreamer](#) 1.18.4 或更新版本。如需詳細資訊，請參閱 [安裝 GStreamer](#)。

在具有 的裝置上 apt，您可以執行下列命令來安裝 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
```

- 每個核心裝置的 AWS IoT SiteWise 資產。此 AWS IoT SiteWise 資產代表核心裝置。如需如何建立此資產的詳細資訊，請參閱 [AWS IoT TwinMaker 使用者指南中的 TwinMaker 影片整合](#)。AWS IoT TwinMaker
- 您連接至每個核心裝置之每個 IP 攝影機的 AWS IoT SiteWise 資產。這些 AWS IoT SiteWise 資產代表將視訊串流到每個核心裝置的攝影機。每個攝影機的資產都必須與連接到攝影機的核心裝置資產相關聯。攝影機資產具有您可以設定的屬性，以指定 Kinesis 影片串流、身分驗證秘密和影片串流參

數。如需如何建立和設定攝影機資產的詳細資訊，請參閱《[AWS IoT TwinMaker 使用者指南](#)》中的 [TwinMaker 視訊整合](#)。AWS IoT TwinMaker

- 每個 IP 攝影機的 AWS Secrets Manager 秘密。此秘密必須定義金鑰/值對，其中金鑰為 RTSPStreamUrl，而值為攝影機的 URL。如果攝影機需要身分驗證，請在此 URL 中包含使用者名稱和密碼。您可以在建立此元件所需的資源時，使用指令碼來建立秘密。如需詳細資訊，請參閱[AWS IoT TwinMaker 使用者指南](#)》中的 [TwinMaker 視訊整合](#)。AWS IoT TwinMaker

您也可以使用 Secrets Manager 主控台和 API 來建立其他秘密。如需詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[建立秘密](#)。

- [Greengrass 權杖交換角色](#)必須允許下列 AWS Secrets Manager AWS IoT SiteWise和 Kinesis Video Streams 動作，如下列範例 IAM 政策所示。

Note

此範例政策允許裝置取得名為 **IPCamera1Url**和 之秘密的值**IPCamera2Url**。當您設定每個 IP 攝影機時，您可以指定包含該攝影機 URL 的秘密。如果攝影機需要身分驗證，您也可以在此 URL 中指定使用者名稱和密碼。核心裝置的字符交換角色必須允許存取每個 IP 攝影機連線的秘密。

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

如果您使用客戶受管 AWS Key Management Service 金鑰來加密秘密，裝置角色也必須允許 kms:Decrypt 動作。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
kinesisvideo. <i>region</i> .amazonaws.com	443	是	將資料上傳至 Kinesis Video Streams。
data.iotsitewise. <i>region</i> .amazonaws.com	443	是	將影片串流中繼資料發佈至

Endpoint	站點	必要	Description
			AWS IoT SiteWise。
secretsmanager. <i>region</i> .amazonaws.com	443	是	將相機 URL 秘密下載至核心裝置。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

下表列出此元件 1.0.0 到 1.0.5 版的相依性。

相依性	相容版本	相依性類型
字符交換服務	>=2.0.3	硬式
串流管理員	>=2.0.9	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

SiteWiseAssetIdForHub

代表此核心裝置的 AWS IoT SiteWise 資產 ID。如需如何建立此資產並使用它與此元件互動的詳細資訊，請參閱《[AWS IoT TwinMaker 使用者指南](#)》中的 [TwinMaker 視訊整合](#)。AWS IoT TwinMaker

Example範例：組態合併更新

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

授權

此元件包含下列第三方軟體/授權：

- [Jetson 任務排程器](#)/Apache 授權 2.0
- [GStreamer 1.x / GNU Lesser General Public License v3.0 的 Java 繫結](#)

Usage

若要設定並與此元件互動，您可以在代表核心裝置及其連線之 IP 攝影機的 AWS IoT SiteWise 資產上設定屬性。您也可以透過 AWS IoT TwinMaker 在 Grafana 儀表板中視覺化視訊串流並與之互動。如需詳細資訊，請參閱[AWS IoT TwinMaker 使用者指南](#)中的 [TwinMaker 視訊整合](#)。AWS IoT TwinMaker

本機日誌檔案

此元件使用以下日誌檔案。

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.0.5	一般錯誤修正與改進。
1.0.4	錯誤修正和改善 修正導致即時上傳停止的問題。
1.0.3	一般錯誤修正與改進。
1.0.1	一般錯誤修正與改進。
1.0.0	初始版本。

另請參閱

- [AWS IoT TwinMaker 使用者指南中的什麼是TwinMaker？](#) AWS IoT TwinMaker
- [AWS IoT TwinMaker 使用者指南中的 TwinMaker 影片整合](#) AWS IoT TwinMaker
- AWS IoT SiteWise 《使用者指南》中的[什麼是 AWS IoT SiteWise？](#)
- AWS IoT SiteWise 《使用者指南》中的[更新屬性值](#)
- 《AWS Secrets Manager 使用者指南》中的[什麼是 AWS Secrets Manager？](#)
- AWS Secrets Manager 《使用者指南》中的[建立和管理秘密](#)

Greengrass CLI

Greengrass CLI 元件 (aws.greengrass.Cli) 提供本機命令列界面，您可以在核心裝置上用來在本機開發和偵錯元件。例如，Greengrass CLI 可讓您在核心裝置上建立本機部署並重新啟動元件。

您可以在安裝 AWS IoT Greengrass 核心軟體時安裝此元件。如需詳細資訊，請參閱[教學課程：入門 AWS IoT Greengrass V2](#)。

Important

我們建議您僅在開發環境中使用此元件，而不是在生產環境中。此元件可讓您存取生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

安裝此元件之後，請執行下列命令來檢視其說明文件。當此元件安裝時，它會greengrass-cli在 ***/greengrass/v2/bin*** 資料夾中將符號連結新增至 。您可以從此路徑執行 Greengrass CLI，或將其新增至您的PATH環境變數，以便在greengrass-cli沒有絕對路徑的情況下執行。

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

例如com.example.HelloWorld，下列命令會重新啟動名為 的元件。

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"
```

如需詳細資訊，請參閱[Greengrass 命令列界面](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 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 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核會在](#)與核相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 您必須獲得使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動的授權。執行下列其中一項操作以使用 Greengrass CLI：
 - 使用執行 AWS IoT Greengrass Core 軟體的系統使用者。
 - 使用具有根或管理許可的使用者。在 Linux 核心裝置上，您可以使用 `sudo` 來取得根許可。
 - 當您部署元件時，請使用您在 `AuthorizedPosixGroups` 或 `AuthorizedWindowsGroups` 組態參數中指定的群組中的系統使用者。如需詳細資訊，請參閱 [Greengrass CLI 元件組態](#)。
- Greengrass CLI 元件支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件 [發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以在此 [AWS IoT Greengrass 主控台](#) 中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.16.0

下表列出此元件 2.16.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	<code>>=2.12.0 <2.17.0</code>	軟式

2.15.1

下表列出此元件 2.15.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.12.0 <2.16.0	軟式

2.15.0

下表列出此元件 2.15.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.12.0 <2.16.0	軟式

2.14.0 – 2.14.3

下表列出此元件 2.14.0 和 2.14.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.12.0 <2.15.0	軟式

2.13.0

下表列出此元件 2.13.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.12.0 <2.14.0	軟式

2.12.0 – 2.12.6

下表列出此元件 2.12.0 至 2.12.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.12.0 <2.13.0	軟式

2.11.0 – 2.11.3

下表列出此元件 2.11.0 到 2.11.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.11.0 <2.12.0	軟式

2.10.0 – 2.10.3

下表列出此元件 2.10.0 到 2.10.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.11.0	軟式

2.9.0 – 2.9.6

下表列出此元件 2.9.0 到 2.9.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.10.0	軟式

2.8.0 – 2.8.1

下表列出此元件 2.8.0 和 2.8.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.9.0	軟式

2.7.0

下表列出此元件 2.7.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.8.0	軟式

2.6.0

下表列出此元件 2.6.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.7.0	軟式

2.5.0 – 2.5.6

下表列出此元件 2.5.0 到 2.5.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.6.0	軟式

2.4.0

下表列出此元件 2.4.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.5.0	軟式

2.3.0

下表列出此元件 2.3.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.4.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.3.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.2.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.1.0	軟式

Note

Greengrass 核的最低相容版本對應至 Greengrass CLI 元件的修補程式版本。

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.5.x - 2.14.x

AuthorizedPosixGroups

(選用) 字串，其中包含以逗號分隔的系統群組清單。您授權這些系統群組使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動。您可以指定群組名稱或群組 IDs。例如，
會group1,1002,group3授權三個系統群組 (group1、1002和 group3) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，則可以使用 Greengrass CLI 做為根使用者 (sudo) 或執行 AWS IoT Greengrass Core 軟體的系統使用者。

AuthorizedWindowsGroups

(選用) 字串，其中包含以逗號分隔的系統群組清單。您授權這些系統群組使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動。您可以指定群組名稱或群組 IDs。例如，
會group1,1002,group3授權三個系統群組 (group1、1002和 group3) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，則可以使用 Greengrass CLI 做為管理員或執行 AWS IoT Greengrass Core 軟體的系統使用者。

Example範例：組態合併更新

下列範例組態指定 授權三個 POSIX 系統群組 (group1、和 group3) 1002和兩個 Windows 使用者群組 (Device Operators 和 QA Engineers) 使用 Greengrass CLI。

```
{  
  "AuthorizedPosixGroups": "group1,1002,group3",  
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"  
}
```

2.4.x - 2.0.x

AuthorizedPosixGroups

(選用) 字串，其中包含以逗號分隔的系統群組清單。您授權這些系統群組使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動。您可以指定群組名稱或群組 IDs。例如，
會group1,1002,group3授權三個系統群組 (group1、1002和 group3) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，則可以使用 Greengrass CLI 做為根使用者 (sudo) 或執行 AWS IoT Greengrass Core 軟體的系統使用者。

Example 範例：組態合併更新

下列範例組態指定 授權三個系統群組 (group1、1002 和 group3) 使用 Greengrass CLI。

```
{  
  "AuthorizedPosixGroups": "group1,1002,group3"  
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.16.1	針對 Greengrass nucleus 2.16.1 版更新版本。
2.16.0	針對 Greengrass nucleus 2.16.0 版更新版本。
2.15.1	針對 Greengrass nucleus 2.15.1 版更新版本。
2.15.0	錯誤修正和改善 • 新增修正，讓身分驗證組態變更時，元件不會重新安裝，但仍會重新啟動。 • 一般錯誤修正與改進。
2.14.3	針對 Greengrass nucleus 2.14.3 版更新版本。
2.14.2	針對 Greengrass nucleus 2.14.2 版更新版本。
2.14.1	針對 Greengrass nucleus 2.14.1 版更新版本。
2.14.0	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 • 在 cli 命令中驗證部署目標參數。
2.13.0	針對 Greengrass nucleus 2.13.0 版更新版本。
2.12.6	針對 Greengrass nucleus 2.12.6 版更新版本。
2.12.5	針對 Greengrass nucleus 2.12.5 版更新版本。
2.12.4	針對 Greengrass nucleus 2.12.4 版更新版本。

版本	變更
2.12.3	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 針對 Greengrass nucleus 2.12.3 版更新版本。
2.12.2	針對 Greengrass nucleus 2.12.2 版更新版本。
2.12.1	針對 Greengrass nucleus 2.12.1 版更新版本。
2.12.0	針對 Greengrass nucleus 2.12.0 版更新版本。
2.11.3	針對 Greengrass nucleus 2.11.3 版更新版本。
2.11.2	針對 Greengrass nucleus 2.11.2 版更新版本。
2.11.1	針對 Greengrass nucleus 2.11.1 版更新版本。
2.11.0	新功能 • 可讓您取消本機部署。 • 可讓您設定本機部署的故障處理政策。 • 改善詳細的部署狀態報告。
2.10.3	針對 Greengrass nucleus 2.10.3 版更新版本。
2.10.2	針對 Greengrass nucleus 2.10.2 版更新版本。
2.10.1	針對 Greengrass nucleus 2.10.1 版更新版本。
2.10.0	針對 Greengrass nucleus 2.10.0 版更新版本。
2.9.6	針對 Greengrass nucleus 2.9.6 版更新版本。
2.9.5	針對 Greengrass nucleus 2.9.5 版更新版本。
2.9.4	針對 Greengrass nucleus 2.9.4 版更新版本。

版本	變更
2.9.3	針對 Greengrass nucleus 2.9.3 版更新版本。
2.9.2	針對 Greengrass nucleus 2.9.2 版更新版本。
2.9.1	針對 Greengrass nucleus 2.9.1 版更新版本。
2.9.0	針對 Greengrass nucleus 2.9.0 版更新版本。
2.8.1	針對 Greengrass nucleus 2.8.1 版更新版本。
2.8.0	針對 Greengrass nucleus 2.8.0 版更新版本。
2.7.0	針對 Greengrass nucleus 2.7.0 版更新版本。
2.6.0	新功能 - 新增對自訂元件的支援，以呼叫 Greengrass CLI 使用的處理程序間通訊 (IPC) 操作。您可以使用這些 IPC 操作來管理本機部署、檢視元件詳細資訊，以及產生可用來登入本機偵錯主控台的密碼。如需詳細資訊，請參閱IPC：管理本機部署和元件。 錯誤修正和改善 - 其他次要修正和改進。
2.5.6	針對 Greengrass nucleus 2.5.6 版更新版本。
2.5.5	針對 Greengrass nucleus 2.5.5 版更新版本。
2.5.4	針對 Greengrass nucleus 2.5.4 版更新版本。
2.5.3	針對 Greengrass nucleus 2.5.3 版更新版本。
2.5.2	針對 Greengrass nucleus 2.5.2 版更新版本。
2.5.1	針對 Greengrass nucleus 2.5.1 版更新版本。

版本	變更
2.5.0	新功能 • 新增對執行 Windows 之核心裝置的支援。 • 新增您可以指定的新AuthorizedWindowsGroups 組態參數，以授權系統群組在 Windows 裝置上使用 Greengrass CLI。 • 新增本機部署的 windowsUser 參數。您可以使用此參數指定要用來在 Windows 核心裝置上執行元件的使用者。
2.4.0	新功能 • 新增對系統資源限制的支援。建立本機部署時，您可以設定每個元件程序可在核心裝置上使用的 CPU 和 RAM 用量上限。如需詳細資訊，請參閱 設定元件的系統資源限制 和 部署建立命令。
2.3.0	針對 Greengrass nucleus 2.3.0 版更新版本。
2.2.0	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.0.5 版更新版本。
2.0.4	針對 Greengrass nucleus 2.0.4 版更新版本。
2.0.3	初始版本。

IP 偵測器

IP 偵測器元件 (aws.greengrass.clientdevices.IPDetector) 會執行下列動作：

- 監控 Greengrass 核心裝置的網路連線資訊。此資訊包括核心裝置的網路端點，以及 MQTT 代理程式操作所在的連接埠。
- 更新 AWS IoT Greengrass 雲端服務中核心裝置的連線資訊。

用戶端裝置可以使用 Greengrass 雲端探索來擷取相關聯的核心裝置的連線資訊。然後，用戶端裝置可以嘗試連線到每個核心裝置，直到成功連線為止。

 Note

用戶端裝置是連線至 Greengrass 核心裝置的本機 IoT 裝置，用於傳送 MQTT 訊息和要處理的資料。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

IP 偵測器元件會將核心裝置的現有連線資訊取代為其偵測到的資訊。由於此元件會移除現有資訊，您可以使用 IP 偵測器元件，或手動管理連線資訊。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.2.x 版本
- 2.1.x
- 2.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。[Greengrass 核心](#)會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- [Greengrass 服務角色](#) 必須與您的 相關聯，AWS 帳戶 並允許 `iot:GetThingShadow` 和 `iot:UpdateThingShadow` 許可。
- 核心裝置的 AWS IoT 政策必須允許 `greengrass:UpdateConnectivityInfo` 許可。如需詳細資訊，請參閱 [AWS IoT 資料平面操作的 政策](#) 及 [支援用戶端裝置的最低 AWS IoT 政策](#)。
- 如果您將核心裝置的 MQTT 代理程式元件設定為使用預設連接埠 8883 以外的連接埠，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告代理程式操作所在的連接埠。
- 如果您有複雜的網路設定，IP 偵測器元件可能無法識別用戶端裝置可以連線到核心裝置的端點。如果 IP 偵測器元件無法管理端點，您必須改為手動管理核心裝置端點。例如，如果核心裝置位於轉送 MQTT 代理程式連接埠的路由器後方，您必須將路由器的 IP 地址指定為核心裝置的端點。如需詳細資訊，請參閱 [管理核心裝置端點](#)。
- IP 偵測器元件支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件 [發行版本的](#) 相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以在此 [AWS IoT Greengrass 主控台](#) 中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.2.3

下表列出此元件 2.2.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.17.0	軟式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.16.0	軟式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.15.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.14.0	軟式

2.1.8 – 2.1.9

下表列出此元件 2.1.8 和 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.13.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.12.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.11.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.10.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.9.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.8.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.7.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.6.0	軟式

2.1.0 and 2.0.2

下表列出此元件 2.1.0 和 2.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.5.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.4.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.3.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.2.x

defaultPort

(選用) 此元件偵測到 IP 地址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。

預設：8883

includeIPv4LoopbackAddr

(選用) 您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 localhost，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。

預設：false

includeIPv4LinkLocalAddr

(選用) 您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。

預設：false

includeIPv6LoopbackAddr

(選用) 您可以啟用此選項來偵測和報告 IPv6 回送地址。這些是 IP 地址，例如 localhost，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。您必須將 includeIPv4Addr 設定為 false，並將 includeIPv6Addr 設定為 true，才能使用此選項。

預設：false

includeIPv6LinkLocalAddr

(選用) 您可以啟用此選項來偵測和報告 IPv6 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態通訊協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。您必須將 includeIPv4Addrs 設定為 false，並將 includeIPv6Addrs 設定為 true，才能使用此選項。

預設：false

includeIPv4Addrs

(選用) 預設值設定為 true。您可以啟用此選項來發佈核心裝置上找到的 IPv4 地址。

預設：true

includeIPv6Addrs

(選用) 您可以啟用此選項來發佈核心裝置上找到的 IPv6 地址。includeIPv4Addrs 設定為 false 以使用此選項。

預設：false

2.1.x

defaultPort

(選用) 此元件偵測到 IP 地址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。

預設：8883

includeIPv4LoopbackAddrs

(選用) 您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 localhost，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。

預設：false

includeIPv4LinkLocalAddr

(選用) 您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。

預設：false

2.0.x

includeIPv4LoopbackAddr

(選用) 您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 localhost，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。

預設：false

includeIPv4LinkLocalAddr

(選用) 您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。

預設：false

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.2.3	針對 Greengrass nucleus 2.16.0 版更新版本。
2.2.2	針對 Greengrass nucleus 2.15.0 版更新版本。
2.2.1	針對 Greengrass nucleus 2.14.0 版更新版本。
2.2.0	針對 Greengrass nucleus 2.13.0 版更新版本。 新功能 • 新增 IPv6 的支援。您現在可以將 IPv6 用於本機傳訊。
2.1.9	錯誤修正和改善 • 調整 IP 取得的步驟，以僅在偵錯日誌層級傳送日誌。
2.1.8	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.2	錯誤修正和改善 • 改善此元件在特定案例中記錄的錯誤訊息。

版本	變更
	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.0	改善項目 - 新增 defaultPort 參數，可讓您使用非預設的 MQTT 代理程式連接埠。 - 更新以讓日誌訊息更清楚。
2.0.2	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.1	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.0	初始版本。

Firehose

Firehose 元件 (aws.greengrass.KinesisFirehose) 透過 Amazon Data Firehose 交付串流發佈資料到目的地，例如 Amazon S3、Amazon Redshift 和 Amazon OpenSearch Service。如需詳細資訊，請參閱 [《Amazon Data Firehose 開發人員指南》中的什麼是 Amazon Data Firehose？](#)。

若要使用此元件發佈至 Kinesis 交付串流，請將訊息發佈至此元件訂閱的主題。根據預設，此元件會訂閱 kinesisfirehose/message 和 kinesisfirehose/message/binary/#[本機發佈/訂閱](#) 主題。您可以在部署此元件時指定其他主題，包括 AWS IoT Core MQTT 主題。

Note

此元件提供與 Firehose 連接器 in AWS IoT Greengrass V1 類似的功能。如需詳細資訊，請參閱 [《AWS IoT Greengrass V1 開發人員指南》中的 Firehose 連接器](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)

- [要求](#)
- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [輸出資料](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是 Lambda 元件 (aws.greengrass.lambda)。 [Greengrass 核](#) 會使用 Lambda [啟動器元件執行此元件的 Lambda](#) 函數。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- [Greengrass 裝置角色](#) 必須允許 firehose:PutRecord 和 firehose:PutRecordBatch 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-name"
      ]
    }
  ]
}
```

您可以動態覆寫此元件輸入訊息承載中的預設交付串流。如果您的應用程式使用此功能，IAM 政策必須包含所有目標串流做為資源。您可以為資源授予精細或條件式存取 (例如，使用萬用字元 * 命名機制)。

- 若要從此元件接收輸出資料，您必須在部署此元件時合併[舊版訂閱路由器元件](#) (aws.greengrass.LegacySubscriptionRouter) 的下列組態更新。此組態會指定此元件發佈回應的主題。

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"
  }
}
}

```

- 以您使用 AWS 區域 的 取代##。
- 以此元件執行的 Lambda 函數版本取代 版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的配方。在[AWS IoT Greengrass 主控台](#)中開啟此元件的詳細資訊頁面，並尋找 Lambda 函數鍵值對。此鍵/值對包含 Lambda 函數的名稱和版本。

Important

每次部署此元件時，都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保針對您部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱[建立部署](#)。

- Firehose 元件支援在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
 - Firehose 元件必須具有firehose.region.amazonaws.com具有 VPC 端點的連線com.amazonaws.region.kinesis-firehose能力。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
firehose. <i>region</i> .amazonaws.com	443	是	將資料上傳至 Firehose。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.16.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.15.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.14.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	^2.0.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.8 - 2.1.0

下表列出此元件 2.0.8 和 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	硬式
Lambda 啟動器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式

相依性	相容版本	相依性類型
字符交換服務	>=1.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

Note

此元件的預設組態包含 Lambda 函數參數。建議您只編輯下列參數，以在裝置上設定此元件。

lambdaParams

包含此元件 Lambda 函數參數的物件。此物件包含下列資訊：

EnvironmentVariables

包含 Lambda 函數參數的物件。此物件包含下列資訊：

DEFAULT_DELIVERY_STREAM_ARN

元件傳送資料之預設 Firehose 交付串流的 ARN。您可以使用輸入訊息承載中的 `delivery_stream_arn` 屬性覆寫目的地串流。

Note

核心裝置角色必須允許對所有目標交付串流執行必要的動作。如需詳細資訊，請參閱[要求](#)。

PUBLISH_INTERVAL

(選用) 在元件將批次資料發佈至 Firehose 之前等待的秒數上限。若要設定元件在接收指標時發佈指標，這表示無需批次處理，請指定 0。

此值最多可達 900 秒。

預設：10 秒

`DELIVERY_STREAM_QUEUE_SIZE`

(選用) 在元件拒絕相同交付串流的新記錄之前，要保留在記憶體中的記錄數目上限。

此值必須至少為 2,000 筆記錄。

預設：5,000 筆記錄

`containerMode`

(選用) 此元件的容器化模式。您可以從以下選項中選擇：

- `NoContainer` – 元件不會在隔離的執行時間環境中執行。
- `GreengrassContainer` – 元件會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

預設：`GreengrassContainer`

`containerParams`

(選用) 包含此元件容器參數的物件。如果您 `GreengrassContainer` 為指定，元件會使用這些參數 `containerMode`。

此物件包含下列資訊：

`memorySize`

(選用) 要配置給元件的記憶體量 (以 KB 為單位)。

預設為 64 MB (65,535 KB)。

`pubsubTopics`

(選用) 物件，其中包含元件訂閱接收訊息的主題。您可以指定每個主題，以及元件是從或 AWS IoT Core 本機發佈/訂閱主題訂閱 MQTT 主題。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

`type`

(選用) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，主題不能包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- IOT_CORE – 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設：PUB_SUB

topic

(選用) 元件訂閱接收訊息的主題。如果您 IotCore 為指定 type，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

Example 範例：組態合併更新（容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 範例：組態合併更新（無容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "NoContainer"
}
```

輸入資料

此元件接受下列主題的串流內容，並將內容傳送至目標交付串流。元件接受兩種類型的輸入資料：

- `kinesisfirehose/message` 主題上的 JSON 資料。
- `kinesisfirehose/message/binary/#` 主題上的二進位資料。

JSON 資料的預設主題（本機發佈/訂閱）：`kinesisfirehose/message`

訊息接受下列屬性。輸入訊息必須是 JSON 格式。

`request`

要傳送到交付串流和目標交付串流 (如果與預設串流不同) 的資料。

類型：`object` 包含下列資訊：

`data`

要傳送到交付串流的資料。

類型：`string`

`delivery_stream_arn`

（選用）目標 Firehose 交付串流的 ARN。指定此屬性以覆寫預設交付串流。

類型：`string`

`id`

請求的任意 ID。使用此屬性將輸入請求映射至輸出回應。當您指定此屬性時，元件會將回應物件中的 `id` 屬性設定為此值。

類型：`string`

Example 範例輸入

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

二進位資料的預設主題（本機發佈/訂閱）：`kinesisfirehose/message/binary/#`

使用此主題傳送包含二進位資料的訊息。元件不會剖析二進位資料。元件會依原樣串流資料。

若要將輸入請求映射到輸出回應，請將訊息主題中的 # 萬用字元換成任意請求 ID。例如，如果您將訊息發佈到 `kinesisfirehose/message/binary/request123`，回應物件中的 `id` 屬性會設為 `request123`。

如果您不想將請求映射到回應，您可以將訊息發佈到 `kinesisfirehose/message/binary/`。請務必包含結尾斜線 (/)。

輸出資料

此元件預設會將回應發佈為下列 MQTT 主題的輸出資料。您必須將此主題指定為 [舊版訂閱路由器元件組態](#) `subject` 中的。如需如何在自訂元件中訂閱此主題訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設主題 (AWS IoT Core MQTT)：`kinesisfirehose/message/status`

Example 範例輸出

回應中包含批次中所傳送之各筆資料記錄的狀態。

```
{
  "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

如果元件偵測到可以重試的錯誤，例如連線錯誤，它會在下一個批次中重試發佈。

本機日誌檔案

此元件使用以下日誌檔案。

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

授權

此元件包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.10	針對 Greengrass nucleus 2.15.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.14.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.13.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.0	新功能 • 新增對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理及讓核心裝置信任 HTTPS 代理。
2.0.8	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

另請參閱

- [《Amazon Data Firehose 開發人員指南》中的什麼是 Amazon Data Firehose？](#)

Lambda 啟動器

Lambda 啟動器元件 (`aws.greengrass.LambdaLauncher`) 會在 AWS IoT Greengrass 核心裝置上啟動和停止 AWS Lambda 函數。此元件也會設定任何容器化，並以您指定的使用者身分執执行程序。

Note

當您將 Lambda 函數元件部署至核心裝置時，部署也會包含此元件。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.0.x

類型

此元件是一般元件 (`aws.greengrass.generic`)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 支援在 VPC 中執行 Lambda 啟動器元件。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

2.0.11 – 2.0.13

下表列出此元件 2.0.11 至 2.0.13 版的相依性。

相依性	相容版本	相依性類型
Lambda 管理員	>=2.0.0 <2.4.0	硬式

2.0.9 – 2.0.10

下表列出此元件 2.0.9 至 2.0.10 版的相依性。

相依性	相容版本	相依性類型
Lambda 管理員	>=2.0.0 <2.3.0	硬式

2.0.4 - 2.0.8

下表列出此元件 2.0.4 至 2.0.8 版的相依性。

相依性	相容版本	相依性類型
Lambda 管理員	>=2.0.0 <2.2.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Lambda 管理員	>=2.0.3 <2.1.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件使用以下日誌檔案。

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑，並將 *lambdaFunctionComponentName* 取代為此元件啟動的 Lambda 函數元件名稱。

```
sudo tail -f /greengrass/v2/logs/lambdaFunctionComponentName.log
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.0.13	錯誤修正和改善 一般錯誤修正與改進。
2.0.12	錯誤修正和改善 修正如果先前的程序未正確停止，Lambda 啟動器可能會擲回錯誤的問題。
2.0.11	支援 Lambda Manager 2.3.0。
2.0.10	錯誤修正和改善 一般錯誤修正與改進。
2.0.9	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.8	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.6	一般效能改進與錯誤修正。
2.0.4	錯誤修正和改善 修正元件未正確傳遞AddGroupOwner 至 Lambda 函數容器的問題。
2.0.3	初始版本。

Lambda 管理員

Lambda Manager 元件 (aws.greengrass.LambdaManager) 會管理在 Greengrass 核心裝置上執行之 AWS Lambda 函數的工作項目和程序間通訊。

Note

當您將 Lambda 函數元件部署至核心裝置時，部署也會包含此元件。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

主題

- [版本](#)
- [作業系統](#)
- [類型](#)
- [需求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

作業系統

此元件只能在 Linux 核心裝置上安裝。

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核會在](#)與核相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

需求

此元件有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 支援 Lambda 管理員元件在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.3.7

下表列出此元件 2.3.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.17.0	軟式

2.3.6

下表列出此元件 2.3.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.16.0	軟式

2.3.5

下表列出此元件 2.3.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.15.0	軟式

2.3.4

下表列出此元件 2.3.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.14.0	軟式

2.3.2 and 2.3.3

下表列出此元件 2.3.2 和 2.3.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.2.10 and 2.3.1

下表列出此元件 2.2.10 版和 2.3.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.2.8 and 2.2.9

下表列出此元件 2.2.8 和 2.2.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.2.5

下表列出此元件 2.2.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.2.4

下表列出此元件 2.2.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.2.1 - 2.2.3

下表列出此元件 2.2.1 至 2.2.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.6.0	軟式

2.1.3 and 2.1.4

下表列出此元件 2.1.3 和 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

logHandlerMode

Note

僅適用於 lambda Manager 2.3.0+ 版

用來選擇要使用的 Lambda 日誌管理員實作。將值設定為 `optimized`，以使用較少的執行緒讀取 lambda 日誌。

getResultTimeoutInSeconds

(選用) Lambda 函數在逾時之前可執行的秒數上限。

預設：60

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

```
/greengrass/v2/logs/greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。`/greengrass/v2` 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.3.7	針對 Greengrass nucleus 2.16.0 版更新版本。
2.3.6	針對 Greengrass nucleus 2.15.0 版更新版本。
2.3.5	錯誤修正和改善 可使用 <code>epoll</code> 而非 <code>nio</code> 來改善效能。
2.3.4	針對 Greengrass nucleus 2.13.0 版更新版本。
2.3.3	錯誤修正和改善 一般錯誤修正與改進。
2.3.2	針對 Greengrass nucleus 2.12.0 版更新版本。
2.3.1	錯誤修正和改善 調整特定錯誤的日誌層級。
2.3.0	新功能 - 日誌處理常式已最佳化，以減少 CPU 負載。將組態選項設定為 <code>logHandlerMode</code>，以使用此功能 <code>optimized</code>。 錯誤修正和改善 - 不再記錄的完整堆疊追蹤 <code>WorkQueueFullException</code>，從而改善日誌和效能。 - 將 Lambda 關閉逾時設定為 15 秒到 300 秒，以防止關閉逾時。

版本	變更
	修正隨需 lambda 在變更組態後可能無法重新啟動的問題。
2.2.11	錯誤修正和改善 修正 Lambda 組態變更時 LegacySubscriptionRouter 組態未更新的問題。
2.2.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.2.9	錯誤修正和改善 修正連接埠號碼因時鐘偏斜而損毀的問題。
2.2.8	針對 Greengrass nucleus 2.10.0 版更新版本。
2.2.7	針對 Greengrass nucleus 2.9.0 版更新版本。
2.2.6	針對 Greengrass nucleus 2.8.0 版更新版本。
2.2.5	新功能 - 在您訂閱本機發佈/訂閱訊息的事件來源中新增對 MQTT 主題萬用字元的支援。 此功能需要 2.6.0 版或更新版本的 Greengrass 核元件。 - 針對 Greengrass nucleus 2.7.0 版更新版本。
2.2.4	針對 Greengrass nucleus 2.6.0 版更新版本。
2.2.3	錯誤修正和改善 修正 Lambda 函數的多個執行個體共用單一 cgroup 的問題。此元件使用 cgroup 來管理 Lambda 函數的資源用量。
2.2.2	錯誤修正和改善 修正鎖定 Lambda 函數元件在特定情況下意外重新啟動的問題。
2.2.1	錯誤修正和改善 變更此元件的 Greengrass 核相依 性版本限制，以修正相依性解析問題。

版本	變更
2.2.0	錯誤修正和改善 - 修正 Lambda 函數在重新啟動後無法寫入日誌的問題。 - 修正當主題中有萬用字元時，舊版訂閱路由器傳送重複訊息的問題。 - 修正非固定 Lambda 函數無法在 中使用 Greengrass 處理程序間通訊 (IPC) 程式庫的問題 AWS IoT Device SDK。
2.1.4	錯誤修正和改善 - 修正導致使用 NodeJS 執行時間的 Lambda 函數僅處理一則訊息的問題。 - 針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

Lambda 執行期

Lambda 執行期元件 (`aws.greengrass.LambdaRuntimes`) 提供 Greengrass 核心裝置用來執行 AWS Lambda 函數的執行期。

Note

當您將 Lambda 函數元件部署至核心裝置時，部署也會包含此元件。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

主題

- [版本](#)
- [Type](#)

- [作業系統](#)
- [要求](#)
- [相依性](#)
- [組態](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱 [Lambda 函數需求](#)。
- 支援在 VPC 中執行 Lambda 執行時間元件。

相依性

此元件沒有任何相依性。

組態

此元件沒有任何組態參數。

本機日誌檔案

此元件不會輸出日誌。

變更記錄

下表說明每個版本的元件的變更。

版本	變更
2.0.9	錯誤修正和改善 使用 Python 3.12 修正語法警告
2.0.8	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

傳統訂閱路由器

舊版訂閱路由器 (`aws.greengrass.LegacySubscriptionRouter`) 會管理 Greengrass 核心裝置上的訂閱。訂閱是 AWS IoT Greengrass V1 的一項功能，定義 Lambda 函數可用於核心裝置上的 MQTT 訊息的主題。如需詳細資訊，請參閱《[AWS IoT Greengrass V1 開發人員指南](#)》中的 [MQTT 訊息工作流程中的受管訂閱](#)。

您可以使用此元件為使用 AWS IoT Greengrass Core SDK 的連接器元件和 Lambda 函數元件啟用訂閱。

Note

只有在您的 Lambda 函數在 AWS IoT Greengrass 核心 SDK 中使用 `publish()` 函數時，才需要舊版訂閱路由器元件。如果您更新 Lambda 函數程式碼以使用 AWS IoT Device SDK V2

中的程序間通訊 (IPC) 界面，則不需要部署舊版訂閱路由器元件。如需詳細資訊，請參閱下列[程序間通訊服務](#)：

- [發佈/訂閱本機訊息](#)
- [發佈/訂閱 AWS IoT Core MQTT 訊息](#)

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能安裝在 Linux 核心裝置上。

要求

此元件有下列需求：

- 舊版訂閱路由器支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.15

下表列出此元件 2.1.15 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.17.0	軟式

2.1.14

下表列出此元件 2.1.14 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.16.0	軟式

2.1.13

下表列出此元件 2.1.13 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.15.0	軟式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.14.0	軟式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v2.1.x

subscriptions

(選用) 在核心裝置上啟用的訂閱。這是物件，其中每個金鑰都是唯一的 ID，而每個值都是定義該連接器訂閱的物件。當您部署使用 AWS IoT Greengrass Core SDK 的 V1 連接器元件或 Lambda 函數時，必須設定訂閱。

每個訂閱物件都包含下列資訊：

id

此訂閱的唯一 ID。此 ID 必須符合此訂閱物件的金鑰。

source

使用 AWS IoT Greengrass 核心 SDK 的 Lambda 函數，發佈 MQTT 訊息到您在 中指定的主題subject。請指定下列其中一項：

- 核心裝置上的 Lambda 函數元件名稱。使用 component：字首指定元件名稱，例如 **component:com.example.HelloWorldLambda**。
- 核心裝置上 Lambda 函數的 Amazon Resource Name (ARN)。

Important

如果 Lambda 函數的版本變更，您必須使用新版本的函數設定訂閱。否則，在版本符合訂閱之前，此元件不會路由訊息。

您必須指定包含要匯入之函數版本的 Amazon Resource Name (ARN)。您不能使用 \$LATEST 之類的版本別名。

若要部署 V1 連接器元件的訂閱，請指定元件的名稱或連接器元件 Lambda 函數的 ARN。

subject

來源和目標可發佈和接收訊息的 MQTT 主題或主題篩選條件。此值支援 +和 #主題萬用字元。

target

針對您在 中指定的主題接收 MQTT 訊息的目標subject。訂閱指定 source函數將 MQTT 訊息發佈到 AWS IoT Core 核心裝置上的 Lambda 函數或 。請指定下列其中一項：

- cloud。source 函數會將 MQTT 訊息發佈至 AWS IoT Core。
- 核心裝置上的 Lambda 函數元件名稱。使用 component：字首指定元件名稱，例如 **component:com.example.HelloWorldLambda**。
- 核心裝置上 Lambda 函數的 Amazon Resource Name (ARN)。

Important

如果 Lambda 函數的版本變更，您必須使用新版本的函數設定訂閱。否則，在版本符合訂閱之前，此元件不會路由訊息。

您必須指定包含要匯入之函數版本的 Amazon Resource Name (ARN)。您不能使用 \$LATEST 之類的版本別名。

預設：無訂閱

Example範例組態更新（定義訂閱 AWS IoT Core）

下列範例指定 com.example.HelloWorldLambda Lambda 函數元件在hello/world主題 AWS IoT Core 上將 MQTT 訊息發佈至。

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example範例組態更新（定義另一個 Lambda 函數的訂閱）

下列範例指定 com.example.HelloWorldLambda Lambda 函數元件將 MQTT 訊息發佈至hello/world主題上的 com.example.MessageRelay Lambda 函數元件。

```
{
  "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

(選用) 在核心裝置上啟用的訂閱。這是物件，其中每個金鑰都是唯一的 ID，而每個值都是定義該連接器訂閱的物件。當您部署使用 AWS IoT Greengrass Core SDK 的 V1 連接器元件或 Lambda 函數時，必須設定訂閱。

每個訂閱物件都包含下列資訊：

id

此訂閱的唯一 ID。此 ID 必須符合此訂閱物件的金鑰。

source

使用 AWS IoT Greengrass 核心 SDK 的 Lambda 函數，發佈 MQTT 訊息到您在 中指定的主題subject。指定下列內容：

- 核心裝置上 Lambda 函數的 Amazon Resource Name (ARN)。

Important

如果 Lambda 函數的版本變更，您必須使用新版本的函數設定訂閱。否則，在版本符合訂閱之前，此元件不會路由訊息。

您必須指定包含要匯入之函數版本的 Amazon Resource Name (ARN)。您不能使用 \$LATEST 之類的版本別名。

若要部署 V1 連接器元件的訂閱，請指定連接器元件 Lambda 函數的 ARN。

subject

來源和目標可發佈和接收訊息的 MQTT 主題或主題篩選條件。此值支援 + 和 # 主題萬用字元。

target

針對您在 中指定的主題接收 MQTT 訊息的目標subject。訂閱指定 source 函數將 MQTT 訊息發佈到 AWS IoT Core 核心裝置上的 Lambda 函數或 。請指定下列其中一項：

- cloud。source 函數會將 MQTT 訊息發佈至 AWS IoT Core。
- 核心裝置上 Lambda 函數的 Amazon Resource Name (ARN)。

Important

如果 Lambda 函數的版本變更，您必須使用新版本的函數設定訂閱。否則，在版本符合訂閱之前，此元件不會路由訊息。

您必須指定包含要匯入之函數版本的 Amazon Resource Name (ARN)。您不能使用 \$LATEST 之類的版本別名。

預設：無訂閱

Example範例組態更新（定義訂閱 AWS IoT Core）

下列範例指定 Greengrass_HelloWorld 函數在 hello/world 主題 AWS IoT Core 上將 MQTT 訊息發佈至。

```
"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範例組態更新（定義另一個 Lambda 函數的訂閱）

下列範例指定 Greengrass_HelloWorld 函數將 MQTT 訊息發佈至 hello/world 主題 Greengrass_MessageRelay 上的。

```
"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"
  }
}
```

本機日誌檔案

此元件不會輸出日誌。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.15	針對 Greengrass nucleus 2.16.0 版更新版本。
2.1.14	針對 Greengrass nucleus 2.15.0 版更新版本。
2.1.13	針對 Greengrass nucleus 2.14.0 版更新版本。
2.1.12	針對 Greengrass nucleus 2.13.0 版更新版本。
2.1.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.7.0 版更新版本。

版本	變更
2.1.5	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	錯誤修正和改善 • 新增支援以指定元件名稱，而不是 source 和 ARNstarget。如果您指定訂閱的元件名稱，則不需要在每次 Lambda 函數版本變更時重新設定訂閱。
2.0.3	初始版本。

本機偵錯主控台

本機偵錯主控台元件 (aws.greengrass.LocalDebugConsole) 提供本機儀表板，可顯示 AWS IoT Greengrass 核心裝置及其元件的相關資訊。您可以使用此儀表板來偵錯核心裝置並管理本機元件。

Important

建議您僅在開發環境中使用此元件，而非生產環境中。此元件可讓您存取您在生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)

- [用途](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 您可以使用使用者名稱和密碼登入儀表板。debug系統會為您提供使用者名稱，即 `debug`。您必須使用 AWS IoT Greengrass CLI 來建立臨時密碼，以使用核心裝置上的儀表板對您進行身分驗證。您必

須能夠使用 AWS IoT Greengrass CLI 來使用本機偵錯主控台。如需詳細資訊，請參閱 [Greengrass CLI 要求](#)。如需如何產生密碼和登入的詳細資訊，請參閱[本機偵錯主控台元件用量](#)。

- 本機偵錯主控台元件支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.4.6

下表列出此元件 2.4.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.17.0	硬式
Greengrass CLI	>=2.10.0 <2.17.0	硬式

2.4.5

下表列出此元件 2.4.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.16.0	硬式
Greengrass CLI	>=2.10.0 <2.16.0	硬式

2.4.4

下表列出此元件 2.4.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.15.0	硬式
Greengrass CLI	>=2.10.0 <2.15.0	硬式

2.4.3

下表列出此元件 2.4.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.14.0	硬式
Greengrass CLI	>=2.10.0 <2.14.0	硬式

2.4.1 – 2.4.2

下表列出此元件 2.4.1 至 2.4.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.13.0	硬式
Greengrass CLI	>=2.10.0 <2.13.0	硬式

2.4.0

下表列出此元件 2.4.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.12.0	硬式
Greengrass CLI	>=2.10.0 <2.12.0	硬式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.10.0 <2.12.0	硬式
Greengrass CLI	>=2.10.0 <2.12.0	硬式

2.2.9

下表列出此元件 2.2.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.12.0	硬式
Greengrass CLI	>=2.1.0 <2.12.0	硬式

2.2.8

下表列出此元件 2.2.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.11.0	硬式
Greengrass CLI	>=2.1.0 <2.11.0	硬式

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.10.0	硬式

相依性	相容版本	相依性類型
Greengrass CLI	>=2.1.0 <2.10.0	硬式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.9.0	硬式
Greengrass CLI	>=2.1.0 <2.9.0	硬式

2.2.5

下表列出此元件 2.2.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.8.0	硬式
Greengrass CLI	>=2.1.0 <2.8.0	硬式

2.2.4

下表列出此元件 2.2.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.7.0	硬式
Greengrass CLI	>=2.1.0 <2.7.0	硬式

2.2.3

下表列出此元件 2.2.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.6.0	硬式
Greengrass CLI	>=2.1.0 <2.6.0	硬式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.5.0	硬式
Greengrass CLI	>=2.1.0 <2.5.0	硬式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.4.0	硬式
Greengrass CLI	>=2.1.0 <2.4.0	硬式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.3.0	硬式
Greengrass CLI	>=2.1.0 <2.3.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.2.0	硬式
Greengrass CLI	>=2.1.0 <2.2.0	硬式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	軟式
Greengrass CLI	>=2.0.3 <2.1.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v2.1.x - v2.4.x

httpsEnabled

(選用) 您可以為本機偵錯主控台啟用 HTTPS 通訊。如果您啟用 HTTPS 通訊，本機偵錯主控台會建立自我簽署憑證。Web 瀏覽器會顯示使用自我簽署憑證的網站的安全警告，因此您必須手動驗證憑證。然後，您可以略過警告。如需詳細資訊，請參閱[用途](#)。

預設：true

port

(選用) 提供本機偵錯主控台的連接埠。

預設：1441

websocketPort

(選用) 用於本機偵錯主控台的 Websocket 連接埠。

預設：1442

bindHostname

(選用) 用於本機偵錯主控台的主機名稱。

如果您在 [Docker 容器中執行 AWS IoT Greengrass Core 軟體](#)，請將此參數設定為 0.0.0.0，以便您可以在 Docker 容器外部開啟本機偵錯主控台。

預設：localhost

Example範例：組態合併更新

下列範例組態指定 在非預設連接埠上開啟本機偵錯主控台，並停用 HTTPS。

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

port

(選用) 提供本機偵錯主控台的連接埠。

預設：1441

websocketPort

(選用) 用於本機偵錯主控台的 Websocket 連接埠。

預設：1442

bindHostname

(選用) 用於本機偵錯主控台的主機名稱。

如果您在 [Docker 容器中執行 AWS IoT Greengrass Core 軟體](#)，請將此參數設定為 0.0.0.0，以便您可以在 Docker 容器外部開啟本機偵錯主控台。

預設：localhost

Example 範例：組態合併更新

下列範例組態指定 在非預設連接埠上開啟本機偵錯主控台。

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

用途

若要使用本機偵錯主控台，請從 Greengrass CLI 建立工作階段。當您建立工作階段時，Greengrass CLI 會提供使用者名稱和密碼，可用來登入本機偵錯主控台。

請依照這些指示，在核心裝置或開發電腦上開啟本機偵錯主控台。

v2.1.x - v2.4.x

在 2.1.0 版和更新版本中，本機偵錯主控台預設使用 HTTPS。啟用 HTTPS 時，本機偵錯主控台會建立自我簽署憑證來保護連線。由於此自我簽署憑證，當您開啟本機偵錯主控台時，網頁瀏覽器會顯示安全警告。當您使用 Greengrass CLI 建立工作階段時，輸出會包含憑證的指紋，因此您可以驗證憑證是否合法且連線安全。

您可以停用 HTTPS。如需詳細資訊，請參閱[本機偵錯主控台組態](#)。

開啟本機偵錯主控台

1. （選用）若要在開發電腦上檢視本機偵錯主控台，您可以透過 SSH 轉送主控台的連接埠。不過，您必須先在核心裝置的 SSH 組態檔案中啟用 AllowTcpForwarding 選項。此選項預設為啟用。在開發電腦上執行下列命令，以在開發電腦上檢視位於 localhost:1441 的儀表板。

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

Note

您可以從 1441 和 變更預設連接埠 1442。如需詳細資訊，請參閱 [本機偵錯主控台組態](#)。

2. 建立工作階段以使用本機偵錯主控台。當您建立工作階段時，會產生用於驗證的密碼。本機偵錯主控台需要密碼來提高安全性，因為您可以使用此元件來檢視重要資訊，並在核心裝置上執行操作。如果您在元件組態中啟用 HTTPS，本機偵錯主控台也會建立憑證來保護連線。HTTPS 預設為啟用。

使用 AWS IoT Greengrass CLI 建立工作階段。此命令會產生隨機的 43 個字元密碼，會在 8 小時後過期。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass V2 根資料夾的路徑。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

如果您已將本機偵錯主控台設定為使用 HTTPS，命令輸出看起來像下列範例。當您開啟本機偵錯主控台時，您可以使用憑證指紋來驗證連線是否安全。

```
Username: debug
Password: bEDp3M0Hdj8ou2w5de_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
```

除錯檢視元件會建立持續 8 小時的工作階段。之後，您必須產生新密碼，才能再次檢視本機偵錯主控台。

3. 開啟並登入儀表板。如果您透過 SSH 轉送連接埠，請在 Greengrass 核心裝置上或在開發電腦上檢視儀表板。執行以下任意一項：

- 如果您在本機偵錯主控台中啟用 HTTPS，這是預設設定，請執行下列動作：
 - a. 如果您透過 SSH 轉送連接埠，`https://localhost:1441`請在核心裝置或開發電腦上開啟。

您的瀏覽器可能會顯示有關無效安全憑證的安全警告。

- b. 如果您的瀏覽器顯示安全性警告，請確認憑證是否合法，並略過安全性警告。請執行下列操作：
 - i. 尋找憑證的 SHA-256 或 SHA-1 指紋，並確認其符合 `get-debug-password` 命令先前列印的 SHA-256 或 SHA-1 指紋。您的瀏覽器可能會提供一個或兩個指紋。請參閱瀏覽器的文件，以檢視憑證及其指紋。在某些瀏覽器中，憑證指紋稱為指紋。

 Note

如果憑證指紋不相符，請前往 [Step 2](#) 建立新的工作階段。如果憑證指紋仍然不相符，您的連線可能不安全。

- ii. 如果憑證指紋相符，請略過瀏覽器的安全性警告，以開啟本機偵錯主控台。請參閱瀏覽器的文件以略過瀏覽器安全警告。
- c. 使用 `get-debug-password` 命令先前列印的使用者名稱和密碼登入網站。

本機偵錯主控台隨即開啟。

- d. 如果本機偵錯主控台顯示錯誤，指出由於 TLS 交握失敗而無法連線至 WebSocket，您必須略過 WebSocket URL 的自我簽署安全警告。

 Error connecting to WebSocket

The connection was closed due to a failure to perform a TLS handshake

Try opening <https://localhost:1442> and bypass any warnings, then reload this page. The WebSocket connection uses the same certificate as this page.

請執行下列操作：

- i. 在開啟本機偵錯主控台的相同瀏覽器<https://localhost:1442>中開啟。
- ii. 驗證憑證並略過安全性警告。

在您略過警告之後，您的瀏覽器可能會顯示 HTTP 404 頁面。

- iii. <https://localhost:1441> 再次開啟。

本機偵錯主控台會顯示核心裝置的相關資訊。

- 如果您在本機偵錯主控台中停用 HTTPS，請執行下列動作：
 - a. 在<http://localhost:1441>核心裝置上開啟，或如果您透過 SSH 轉送連接埠，請在開發電腦上開啟它。
 - b. 使用 `get-debug-password` 命令先前列印的使用者名稱和密碼登入網站。

本機偵錯主控台隨即開啟。

v2.0.x

開啟本機偵錯主控台

1. （選用）若要在開發電腦上檢視本機偵錯主控台，您可以透過 SSH 轉送主控台的連接埠。不過，您必須先在核心裝置的 SSH 組態檔案中啟用 `AllowTcpForwarding` 選項。此選項預設為啟用。在開發電腦上執行下列命令，以在開發電腦上檢視位於 `localhost:1441` 的儀表板。

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

Note

您可以從 1441 和 變更預設連接埠 1442。如需詳細資訊，請參閱 [本機偵錯主控台組態](#)。

2. 建立工作階段以使用本機偵錯主控台。當您建立工作階段時，會產生用於驗證的密碼。本機偵錯主控台需要密碼來提高安全性，因為您可以使用此元件來檢視重要資訊，並在核心裝置上執行操作。

使用 AWS IoT Greengrass CLI 建立工作階段。此命令會產生隨機的 43 個字元密碼，會在 8 小時後過期。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass V2 根資料夾的路徑。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

命令輸出看起來像下列範例。

```
Username: debug
Password: bEDp3M0Hdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

除錯檢視元件會建立工作階段 4 小時，然後您必須產生新密碼才能再次檢視本機除錯主控台。

3. 在 `http://localhost:1441` 核心裝置上開啟，或如果您透過 SSH 轉送連接埠，請在開發電腦上開啟它。
4. 使用 `get-debug-password` 命令先前列印的使用者名稱和密碼登入網站。

本機偵錯主控台隨即開啟。

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```


檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.4.6	針對 Greengrass nucleus 2.16.0 版更新版本。
2.4.5	針對 Greengrass nucleus 2.15.0 版更新版本。
2.4.4	針對 Greengrass nucleus 2.14.0 版更新版本。
2.4.3	針對 Greengrass nucleus 2.13.0 版更新版本。 錯誤修正和改善 修正錯誤 <code>STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH</code> 地以每秒百萬位元 (Mbps) 而非每秒位元組 (Bps) 顯示的問題。
2.4.2	錯誤修正和改善 一般錯誤修正與改進。
2.4.1	針對 Greengrass nucleus 2.12.0 版更新版本。
2.4.0	新功能 新增串流管理員偵錯主控台。

版本	變更
2.3.1	針對 Greengrass nucleus 2.11.0 版更新版本。
2.3.0	針對 Greengrass nucleus 2.10.0 版更新版本。 新功能 • 包括 PubSub 和 AWS IoT Core MQTT 偵錯用戶端。
2.2.7	針對 Greengrass nucleus 2.9.0 版更新版本。
2.2.6	針對 Greengrass nucleus 2.8.0 版更新版本。
2.2.5	針對 Greengrass nucleus 2.7.0 版更新版本。
2.2.4	針對 Greengrass nucleus 2.6.0 版更新版本。
2.2.3	錯誤修正和改善 • 修正當元件無法解密保存 SSL 私有金鑰的金鑰存放區時，無法啟動的問題。 • 針對 Greengrass nucleus 2.5.0 版更新版本。
2.2.2	針對 Greengrass nucleus 2.4.0 版更新版本。
2.2.1	針對 Greengrass nucleus 2.3.0 版更新版本。
2.2.0	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	新功能 • 使用 HTTPS 來保護與本機偵錯主控台的連線。HTTPS 預設為啟用。 錯誤修正和改善 • 您可以在組態編輯器中關閉閃爍列訊息。
2.0.3	初始版本。

日誌管理員

Warning

我們建議您升級至 Log Manager 2.3.5 版或更新版本。2.3.5 版可最佳化 Log Manager 組態寫入、降低 IO 操作並改善日誌上傳速度、整體裝置效能，以及可能延長的裝置生命週期。

日誌管理員元件 (`aws.greengrass.LogManager`) 會將日誌從 AWS IoT Greengrass 核心裝置上傳至 Amazon CloudWatch Logs。您可以從 Greengrass 核、其他 Greengrass 元件，以及其他非 Greengrass 元件的應用程式和服務上傳日誌。如需如何在 CloudWatch Logs 和本機檔案系統上監控日誌的詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

當您使用日誌管理員元件寫入 CloudWatch Logs 時，適用下列考量：

- 日誌延遲

日誌管理員元件 2.2.8 版（及更舊版本）只會從輪換的日誌檔案處理和上傳日誌。根據預設，AWS IoT Greengrass 核心軟體會每小時輪換日誌檔案，或在它們為 1,024 KB 之後輪換日誌檔案。因此，日誌管理員元件只會在 AWS IoT Greengrass Core 軟體或 Greengrass 元件寫入超過 1,024 KB 的日誌之後上傳日誌。您可以設定較低的日誌檔案大小限制，讓日誌檔案更頻繁地輪換。這會導致日誌管理員元件更頻繁地將日誌上傳至 CloudWatch Logs。

日誌管理員元件 2.3.0 版（及更新版本）會處理和上傳所有日誌。當您撰寫新的日誌時，日誌管理員 2.3.0 版（及更新版本）會處理並直接上傳該作用中日誌檔案，而不是等待輪換。這表示您可以在 5 分鐘內檢視新日誌。

日誌管理員元件會定期上傳新日誌。根據預設，日誌管理員元件會每 5 分鐘上傳新日誌。您可以設定較低的上傳間隔，讓日誌管理員元件透過設定 `LogPeriodicUploadIntervalSec`，更頻繁地將日誌上傳至 CloudWatch Logs。如需如何設定此定期間隔的詳細資訊，請參閱[組態](#)。

日誌可以近乎即時地從相同的 Greengrass 檔案系統上傳。如果您需要即時觀察日誌，請考慮使用[檔案系統日誌](#)。

Note

如果您使用不同的檔案系統來寫入日誌，日誌管理員會還原為日誌管理員元件 2.2.8 版和更早版本中的行為。如需有關存取檔案系統日誌的資訊，請參閱[存取檔案系統日誌](#)。

- 時鐘扭曲

日誌管理員元件使用標準 Signature 第 4 版簽署程序來建立對 CloudWatch Logs 的 API 請求。如果核心裝置上的系統時間不同步超過 15 分鐘，則 CloudWatch Logs 會拒絕請求。如需詳細資訊，請參閱 AWS 一般參考 中的 [Signature 第 4 版簽署程序](#)。

如需此元件上傳日誌之日誌群組和日誌串流的相關資訊，請參閱 [Usage](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [Usage](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- [Greengrass 裝置角色](#)必須允許 logs:CreateLogGroup、logs:PutLogEvents、logs:CreateLogStream和 logs:DescribeLogStreams動作，如下列範例 IAM 政策所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Note

根據預設，您在安裝 AWS IoT Greengrass Core 軟體時建立的 [Greengrass 裝置角色](#) 包含此範例政策中的許可。

如需詳細資訊，請參閱《Amazon [CloudWatch Logs 使用者指南](#)》中的針對 [CloudWatch Logs 使用身分型政策 \(IAM 政策\)](#)。Amazon CloudWatch

- 日誌管理員元件支援在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
- 日誌管理員元件必須具有logs.region.amazonaws.com具有 VPC 端點的連線com.amazonaws.us-east-1.logs能力。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
logs. <i>region</i> .amazonaws.com	443	否	如果您將日誌寫入 CloudWatch Logs，則為必要項目。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.3.11

下表列出此元件 2.3.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.17.0	軟式

2.3.10

下表列出此元件 2.3.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.16.0	軟式

2.3.9

下表列出此元件 2.3.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.15.0	軟式

2.3.8

下表列出此元件 2.3.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.14.0	軟式

2.3.7

下表列出此元件 2.3.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.13.0	軟式

2.3.5 and 2.3.6

下表列出此元件 2.3.5 和 2.3.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.12.0	軟式

2.3.3 – 2.3.4

下表列出此元件 2.3.3 至 2.3.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.11.0	軟式

2.2.8 – 2.3.2

下表列出此元件 2.2.8 至 2.3.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.10.0	軟式

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.9.0	軟式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.8.0	軟式

2.2.5

下表列出此元件 2.2.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.7.0	軟式

2.2.1 - 2.2.4

下表列出此元件 2.2.1 - 2.2.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.6.0	軟式

2.1.3 and 2.2.0

下表列出此元件 2.1.3 和 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.5.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.4.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.1.0 <2.3.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	$\geq 2.1.0 < 2.2.0$	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	$\geq 2.0.3 < 2.1.0$	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

v2.3.10

logsUploaderConfiguration

(選用) 日誌管理員元件上傳之日誌的組態。此物件包含下列資訊：

systemLogsConfiguration

(選用) AWS IoT Greengrass 核心軟體系統日誌的組態，其中包含來自 [Greengrass 核](#)和[外掛程式元件](#)的日誌。指定此組態，讓日誌管理員元件能夠管理系統日誌。此物件包含下列資訊：

uploadToCloudWatch

(選用) 您可以將系統日誌上傳至 CloudWatch Logs。

預設：false

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。此最低層級僅適用於您將 [Greengrass 核元件](#) 設定為輸出 JSON 格式日誌的情況。若要啟用 JSON 格式日誌，請JSON為[記錄格式](#)參數 () 指定 logging.format。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) Greengrass 系統日誌檔案的總大小上限，以您在 中指定的單位表示 diskSpaceLimitUnit。在 Greengrass 系統日誌檔案的總大小超過此總大小上限之後，AWS IoT Greengrass Core 軟體會刪除最舊的 Greengrass 系統日誌檔案。

此參數等同於 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (totalLogsSizeKB)。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為 Greengrass 系統日誌大小總計上限。

diskSpaceLimitUnit

(選用) 的單位 diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

deleteLogFileAfterCloudUpload

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

componentLogsConfigurationMap

(選用) 核心裝置上元件的日誌組態映射。此映射中的每個 componentName 物件都會定義元件或應用程式的日誌組態。日誌管理員元件會將這些元件日誌上傳至 CloudWatch Logs。

 Important

我們強烈建議每個元件使用單一組態金鑰。您只能將只有一個日誌檔案在使用時主動寫入的一組檔案設為目標logFileRegex。未遵循此建議可能會導致重複的日誌上傳到 CloudWatch。如果您以單一 regex 鎖定多個作用中日誌檔案，建議您升級至 log Manager v2.3.1 或更新版本，並考慮使用[範例組態變更您的組態](#)。

 Note

如果您要從 v2.2.0 之前的日誌管理員版本升級，您可以繼續使用componentLogsConfiguration清單，而不是componentLogsConfigurationMap。不過，我們強烈建議您使用映射格式，以便您可以使用合併和重設更新來修改特定元件的組態。如需componentLogsConfiguration 參數的相關資訊，請參閱此元件 v2.1.x 的組態參數。

componentName

此日誌組態之*componentName*元件或應用程式的日誌組態。您可以指定 Greengrass 元件的名稱或其他值來識別此日誌群組。

每個物件都包含下列資訊：

minimumLogLevel

（選用）要上傳的日誌訊息最低層級。此最低層級僅適用於此元件的日誌使用特定的 JSON 格式時，您可以在 GitHub 上的[AWS IoT Greengrass 記錄模組](#)儲存庫中找到該格式。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) 此元件的所有日誌檔案總大小上限，以您在 中指定的單位表示 diskSpaceLimitUnit。此元件日誌檔案的總大小超過此總大小上限後，AWS IoT Greengrass 核心軟體會刪除此元件最舊的日誌檔案。

此參數與 [Greengrass 核元件的日誌大小限制](#) 參數 (totalLogsSizeKB) 有關。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為此元件的總日誌大小上限。

diskSpaceLimitUnit

(選用) 的單位 diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

logFileDirectoryPath

(選用) 包含此元件日誌檔案之資料夾的路徑。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

預設：*/greengrass/v2/logs*。

logFileRegex

(選用) 規則表達式，指定元件或應用程式使用的日誌檔案名稱格式。日誌管理員元件使用此規則表達式來識別 資料夾中的日誌檔案 logFileDirectoryPath。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

如果您的元件或應用程式輪換日誌檔案，請指定符合輪換日誌檔案名稱的 regex。例如，您可以指定 **hello_world\\\\w*.log** 上傳 Hello World 應用程式的日誌。\\\\w* 模式符合零個或多個單字字元，其中包含英數字元和底線。此 regex 會比對名稱中含和不含時間戳記的日誌檔案。在此範例中，日誌管理員會上傳下列日誌檔案：

- hello_world.log – Hello World 應用程式的最新日誌檔案。
- hello_world_2020_12_15_17_0.log – Hello World 應用程式的較舊日誌檔案。

預設：`componentName\\\\w*.log`，其中 `componentName` 是此日誌組態的元件名稱。

`deleteLogFileAfterCloudUpload`

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：`false`

`multiLineStartPattern`

(選用) 規則表達式，可識別新行上的日誌訊息何時是新的日誌訊息。如果規則表達式不符合新行，日誌管理員元件會將新行附加到上一行的日誌訊息。

根據預設，日誌管理員元件會檢查該行是否以空格字元開頭，例如標籤或空格。如果沒有，日誌管理員會將該行視為新的日誌訊息處理。否則，它會將該行附加到目前的日誌訊息。此行為可確保日誌管理員元件不會分割跨越多行的訊息，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選用) 以秒為單位的期間，日誌管理員元件會檢查要上傳的新日誌檔案。

預設：`300`(5 分鐘)

下限：`0.000001`(1 微秒)

`updateToTlogIntervalSec`

(選用) 以秒為單位的期間，表示 nucleus 將 Amazon CloudWatch Events 日誌上傳事件詳細資訊寫入本機交易日誌 () 的頻率 `config.tlog`。預設為 中指定的值 `periodicUploadIntervalSec`。您可以修改此參數來增加寫入間隔。

預設：`periodicUploadIntervalSec`

下限：`periodicUploadIntervalSec`

`deprecatedVersionSupport`

指出日誌管理員是否應使用日誌管理員 v2.3.5 中引入的記錄速度改進。將值設定為 `false` 以使用改進。

當您從日誌管理員 v2.3.1 或更早的重複日誌項目升級 `false` 時，如果您將此值設定為 `true`，則可能會上傳重複的日誌項目。

預設值為 true。

Example範例：組態合併更新

下列範例組態指定 將系統日誌和com.example.HelloWorld元件日誌上傳至 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範例：使用 Log Manager v2.3.1 上傳多個作用中日誌檔案的組態

如果您想要以多個作用中日誌檔案為目標，建議使用下列範例組態。此範例組態會指定您要上傳至 CloudWatch 的作用中日誌檔案。使用此組態範例組態也會上傳任何符合 的輪換檔案logFileRegex。日誌管理員 v2.3.1 支援此範例組態。

```
{
  "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

(選用) 日誌管理員元件上傳之日誌的組態。此物件包含下列資訊：

systemLogsConfiguration

(選用) AWS IoT Greengrass 核心軟體系統日誌的組態，其中包含來自 [Greengrass 核](#)和[外掛程式元件](#)的日誌。指定此組態，讓日誌管理員元件能夠管理系統日誌。此物件包含下列資訊：

uploadToCloudWatch

(選用) 您可以將系統日誌上傳至 CloudWatch Logs。

預設：false

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。只有在您將 [Greengrass 核元件](#)設定為輸出 JSON 格式日誌時，才會套用此最低層級。若要啟用 JSON 格式日誌，請JSON為[記錄格式](#)參數()指定 logging.format。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) Greengrass 系統日誌檔案的總大小上限，以您在 中指定的單位表示diskSpaceLimitUnit。在 Greengrass 系統日誌檔案的總大小超過此總大小上限之後，AWS IoT Greengrass Core 軟體會刪除最舊的 Greengrass 系統日誌檔案。

此參數等同於 [Greengrass 核心元件的日誌大小限制](#) 參數 (totalLogsSizeKB)。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為 Greengrass 系統日誌大小總計上限。

diskSpaceLimitUnit

(選用) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

deleteLogFileAfterCloudUpload

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

componentLogsConfigurationMap

(選用) 核心裝置上元件的日誌組態映射。此映射中的每個componentName物件都會定義元件或應用程式的日誌組態。日誌管理員元件會將這些元件日誌上傳至 CloudWatch Logs。

Important

我們強烈建議每個元件使用單一組態金鑰。您只能將只有一個日誌檔案在使用時主動寫入的一組檔案設為目標logFileRegex。未遵循此建議可能會導致重複的日誌上傳到 CloudWatch。如果您以單一 regex 鎖定多個作用中日誌檔案，建議您升級至 log Manager v2.3.1 或更新版本，並考慮使用[範例組態變更您的組態](#)。

Note

如果您要從 v2.2.0 之前的日誌管理員版本升級，您可以繼續使用componentLogsConfiguration清單，而不是

`componentLogsConfigurationMap`。不過，我們強烈建議您使用映射格式，以便您可以使用合併和重設更新來修改特定元件的組態。如需 `componentLogsConfiguration` 參數的相關資訊，請參閱此元件 v2.1.x 的組態參數。

componentName

此日誌組態之 *componentName* 元件或應用程式的日誌組態。您可以指定 Greengrass 元件的名稱或其他值來識別此日誌群組。

每個物件都包含下列資訊：

`minimumLogLevel`

(選用) 要上傳的日誌訊息最低層級。此最低層級僅適用於此元件的日誌使用特定的 JSON 格式時，您可以在 GitHub 上的 [AWS IoT Greengrass 記錄模組](#) 儲存庫中找到該格式。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

`diskSpaceLimit`

(選用) 此元件的所有日誌檔案總大小上限，以您在 中指定的單位表示 `diskSpaceLimitUnit`。此元件日誌檔案的總大小超過此總大小上限後，AWS IoT Greengrass 核心軟體會刪除此元件最舊的日誌檔案。

此參數與 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (`totalLogsSizeKB`) 有關。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為此元件的總日誌大小上限。

`diskSpaceLimitUnit`

(選用) 的單位 `diskSpaceLimit`。您可以從以下選項中選擇：

- KB – KB
- MB – MB

- GB – GB

預設：KB

`logFileDirectoryPath`

(選用) 包含此元件日誌檔案之資料夾的路徑。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

預設：`/greengrass/v2/logs`。

`logFileRegex`

(選用) 規則表達式，指定元件或應用程式使用的日誌檔案名稱格式。日誌管理員元件使用此規則表達式來識別 資料夾中的日誌檔案 `logFileDirectoryPath`。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

如果您的元件或應用程式輪換日誌檔案，請指定符合輪換日誌檔案名稱的 regex。例如，您可以指定 `hello_world\\\\w*.log` 上傳 Hello World 應用程式的日誌。\\\\w* 模式符合零個或多個單字元，其中包含英數字元和底線。此 regex 會比對名稱中含和不含時間戳記的日誌檔案。在此範例中，日誌管理員會上傳下列日誌檔案：

- `hello_world.log` – Hello World 應用程式的最新日誌檔案。
- `hello_world_2020_12_15_17_0.log` – Hello World 應用程式的較舊日誌檔案。

預設：`componentName\\\\w*.log`，其中 `componentName` 是此日誌組態的元件名稱。

`deleteLogFileAfterCloudUpload`

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：`false`

`multiLineStartPattern`

(選用) 規則表達式，可識別新行上的日誌訊息何時是新的日誌訊息。如果規則表達式不符合新行，日誌管理員元件會將新行附加到上一行的日誌訊息。

根據預設，日誌管理員元件會檢查該行是否以空格字元開頭，例如標籤或空格。如果沒有，日誌管理員會將該行視為新的日誌訊息處理。否則，它會將該行附加到目前的日誌訊息。此行為可確保日誌管理員元件不會分割跨越多行的訊息，例如堆疊追蹤。

periodicUploadIntervalSec

(選用) 以秒為單位的期間，日誌管理員元件會檢查要上傳的新日誌檔案。

預設：300(5 分鐘)

下限：0.000001(1 微秒)

deprecatedVersionSupport

指出日誌管理員是否應使用日誌管理員 v2.3.5 中引入的記錄速度改進。將值設定為 false 以使用改進。

當您從日誌管理員 v2.3.1 或更早的重複日誌項目升級 false 時，如果您將此值設定為 true，則可能會上傳重複的日誌項目。

預設值為 true。

Example 範例：組態合併更新

下列範例組態指定 將系統日誌和 com.example.HelloWorld 元件日誌上傳至 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範例：使用 Log Manager v2.3.1 上傳多個作用中日誌檔案的組態

如果您想要以多個作用中日誌檔案為目標，建議使用以下範例組態。此範例組態會指定您要上傳至 CloudWatch 的作用中日誌檔案。使用此組態範例組態也會上傳任何符合的輪換檔案 `logFileRegex`。日誌管理員 v2.3.1 支援此範例組態。

```

{
  "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

(選用) 日誌管理員元件上傳之日誌的組態。此物件包含下列資訊：

systemLogsConfiguration

(選用) AWS IoT Greengrass 核心軟體系統日誌的組態，其中包含來自 [Greengrass 核心和外掛程式元件](#) 的日誌。指定此組態，讓日誌管理員元件能夠管理系統日誌。此物件包含下列資訊：

uploadToCloudWatch

(選用) 您可以將系統日誌上傳至 CloudWatch Logs。

預設：false

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。只有在您將 [Greengrass 核元件](#) 設定為輸出 JSON 格式日誌時，才會套用此最低層級。若要啟用 JSON 格式日誌，請將 `JSON` 為 [記錄格式](#) 參數 () 指定 `logging.format`。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) Greengrass 系統日誌檔案的總大小上限，以您在 `diskSpaceLimitUnit` 中指定的單位表示。在 Greengrass 系統日誌檔案的總大小超過此總大小上限之後，AWS IoT Greengrass Core 軟體會刪除最舊的 Greengrass 系統日誌檔案。

此參數等同於 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (`totalLogsSizeKB`)。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為 Greengrass 系統日誌大小總計上限。

diskSpaceLimitUnit

(選用) 的單位 `diskSpaceLimit`。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

deleteLogFileAfterCloudUpload

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

componentLogsConfigurationMap

(選用) 核心裝置上元件的日誌組態映射。此映射中的每個 `componentName` 物件都會定義元件或應用程式的日誌組態。日誌管理員元件會將這些元件日誌上傳至 CloudWatch Logs。

Important

我們強烈建議每個元件使用單一組態金鑰。您只能將只有一個日誌檔案在使用時主動寫入的一組檔案設為目標 `logFileRegex`。未遵循此建議可能會導致重複的日誌上傳到 CloudWatch。如果您以單一 `regex` 鎖定多個作用中日誌檔案，建議您升級至 log Manager v2.3.1，並考慮使用[範例組態變更您的組態](#)。

Note

如果您要從 v2.2.0 之前的日誌管理員版本升級，您可以繼續使用 `componentLogsConfiguration` 清單，而不是 `componentLogsConfigurationMap`。不過，我們強烈建議您使用映射格式，以便您可以使用合併和重設更新來修改特定元件的組態。如需 `componentLogsConfiguration` 參數的相關資訊，請參閱此元件 v2.1.x 的組態參數。

componentName

此日誌組態之 *componentName* 元件或應用程式的日誌組態。您可以指定 Greengrass 元件的名稱或其他值來識別此日誌群組。

每個物件都包含下列資訊：

`minimumLogLevel`

(選用) 要上傳的日誌訊息最低層級。此最低層級僅適用於此元件的日誌使用特定的 JSON 格式時，您可以在 GitHub 上的[AWS IoT Greengrass 記錄模組](#)儲存庫中找到該格式。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO

- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) 您在 中指定的單位中，此元件所有日誌檔案的總大小上限diskSpaceLimitUnit。此元件日誌檔案的總大小超過此總大小上限後，AWS IoT Greengrass 核心軟體會刪除此元件最舊的日誌檔案。

此參數與 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (totalLogsSizeKB) 有關。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為此元件的總日誌大小上限。

diskSpaceLimitUnit

(選用) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

logFileDirectoryPath

(選用) 包含此元件日誌檔案之資料夾的路徑。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

預設：*/greengrass/v2/logs*。

logFileRegex

(選用) 規則表達式，指定元件或應用程式使用的日誌檔案名稱格式。日誌管理員元件使用此規則表達式來識別 資料夾中的日誌檔案logFileDirectoryPath。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

如果您的元件或應用程式輪換日誌檔案，請指定符合輪換日誌檔案名稱的 regex。例如，您可以指定 **hello_world\\\\\\\\w*.log**上傳 Hello World 應用程式的日誌。\\\\

`\w*` 模式符合零個或多個單字字元，其中包含英數字元和底線。此 regex 會比對名稱中具有和沒有時間戳記的日誌檔案。在此範例中，日誌管理員會上傳下列日誌檔案：

- `hello_world.log` – Hello World 應用程式的最新日誌檔案。
- `hello_world_2020_12_15_17_0.log` – Hello World 應用程式的較舊日誌檔案。

預設：`componentName\\\\\\w*.log`，其中 `componentName` 是此日誌組態的元件名稱。

`deleteLogFileAfterCloudUpload`

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：`false`

`multiLineStartPattern`

(選用) 規則表達式，可識別新行上的日誌訊息何時是新的日誌訊息。如果規則表達式不符合新行，日誌管理員元件會將新行附加到前一行的日誌訊息。

根據預設，日誌管理員元件會檢查該行是否以空格字元開頭，例如標籤或空格。如果沒有，日誌管理員會將該行視為新的日誌訊息處理。否則，它會將該行附加到目前的日誌訊息。此行為可確保日誌管理員元件不會分割跨越多行的訊息，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選用) 以秒為單位的期間，日誌管理員元件會檢查要上傳的新日誌檔案。

預設：`300`(5 分鐘)

最小值：`0.000001`(1 微秒)

Example 範例：組態合併更新

下列範例組態指定 將系統日誌和 `com.example.HelloWorld` 元件日誌上傳至 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範例：使用 Log Manager v2.3.1 上傳多個作用中日誌檔案的組態

如果您想要以多個作用中日誌檔案為目標，建議使用以下範例組態。此範例組態會指定您要上傳至 CloudWatch 的作用中日誌檔案。使用此組態範例組態也會上傳任何符合的輪換檔案 `logFileRegex`。日誌管理員 v2.3.1 支援此範例組態。

```

{
  "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

(選用) 日誌管理員元件上傳之日誌的組態。此物件包含下列資訊：

systemLogsConfiguration

(選用) AWS IoT Greengrass 核心軟體系統日誌的組態，其中包含來自 [Greengrass 核](#)和[外掛程式元件](#)的日誌。指定此組態，讓日誌管理員元件能夠管理系統日誌。此物件包含下列資訊：

uploadToCloudWatch

(選用) 您可以將系統日誌上傳至 CloudWatch Logs。

預設：false

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。只有在您將 [Greengrass 核元件](#)設定為輸出 JSON 格式日誌時，才會套用此最低層級。若要啟用 JSON 格式日誌，請JSON為[記錄格式](#)參數() 指定 logging.format。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) Greengrass 系統日誌檔案的總大小上限，以您在 中指定的單位表示diskSpaceLimitUnit。在 Greengrass 系統日誌檔案的總大小超過此總大小上限之後，AWS IoT Greengrass Core 軟體會刪除最舊的 Greengrass 系統日誌檔案。

此參數等同於 [Greengrass 核元件](#)的[日誌大小限制](#)參數 (totalLogsSizeKB)。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為 Greengrass 系統日誌大小總計上限。

diskSpaceLimitUnit

(選用) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

`deleteLogFileAfterCloudUpload`

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

`componentLogsConfigurationMap`

(選用) 核心裝置上元件的日誌組態映射。此映射中的每個 `componentName` 物件都會定義元件或應用程式的日誌組態。日誌管理員元件會將這些元件日誌上傳至 CloudWatch Logs。

 Note

如果您要從 v2.2.0 之前的日誌管理員版本升級，您可以繼續使用 `componentLogsConfiguration` 清單，而不是 `componentLogsConfigurationMap`。不過，我們強烈建議您使用映射格式，以便您可以使用合併和重設更新來修改特定元件的組態。如需 `componentLogsConfiguration` 參數的相關資訊，請參閱此元件 v2.1.x 的組態參數。

componentName

此日誌組態之 *componentName* 元件或應用程式的日誌組態。您可以指定 Greengrass 元件的名稱或其他值來識別此日誌群組。

每個物件都包含下列資訊：

`minimumLogLevel`

(選用) 要上傳的日誌訊息最低層級。此最低層級僅適用於此元件的日誌使用特定的 JSON 格式時，您可以在 GitHub 上的 [AWS IoT Greengrass 記錄模組](#) 儲存庫中找到該格式。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) 此元件的所有日誌檔案總大小上限，以您在 中指定的單位表示 diskSpaceLimitUnit。此元件日誌檔案的總大小超過此總大小上限後，AWS IoT Greengrass 核心軟體會刪除此元件最舊的日誌檔案。

此參數與 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (totalLogsSizeKB) 有關。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為此元件的總日誌大小上限。

diskSpaceLimitUnit

(選用) 的單位 diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

logFileDirectoryPath

(選用) 包含此元件日誌檔案的資料夾路徑。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

預設：/greengrass/v2/logs。

logFileRegex

(選用) 規則表達式，指定元件或應用程式使用的日誌檔案名稱格式。日誌管理員元件使用此規則表達式來識別 資料夾中的日誌檔案 logFileDirectoryPath。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

如果您的元件或應用程式輪換日誌檔案，請指定符合輪換日誌檔案名稱的 regex。例如，您可以指定 **hello_world\\\\w*.log** 上傳 Hello World 應用程式的日誌。\\\\w* 模式符合零個或多個單字字元，其中包含英數字元和底線。此 regex 會比對名稱中含和不含時間戳記的日誌檔案。在此範例中，日誌管理員會上傳下列日誌檔案：

- hello_world.log – Hello World 應用程式的最新日誌檔案。

- `hello_world_2020_12_15_17_0.log` – Hello World 應用程式的較舊日誌檔案。

預設：`componentName\\\\\\\\w*.log`，其中 `componentName` 是此日誌組態的元件名稱。

`deleteLogFileAfterCloudUpload`

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：`false`

`multiLineStartPattern`

(選用) 規則表達式，可識別新行上的日誌訊息何時是新的日誌訊息。如果規則表達式不符合新行，日誌管理員元件會將新行附加到前一行的日誌訊息。

根據預設，日誌管理員元件會檢查該行是否以空格字元開頭，例如標籤或空格。如果沒有，日誌管理員會將該行視為新的日誌訊息處理。否則，它會將該行附加到目前的日誌訊息。此行為可確保日誌管理員元件不會分割跨越多行的訊息，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選用) 以秒為單位的期間，日誌管理員元件會檢查要上傳的新日誌檔案。

預設：`300`(5 分鐘)

下限：`0.000001`(1 微秒)

Example 範例：組態合併更新

下列範例組態指定 將系統日誌和 `com.example.HelloWorld` 元件日誌上傳至 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

(選用) 日誌管理員元件上傳之日誌的組態。此物件包含下列資訊：

systemLogsConfiguration

(選用) AWS IoT Greengrass 核心軟體系統日誌的組態，其中包含來自 [Greengrass 核](#)和[外掛程式元件](#)的日誌。指定此組態，讓日誌管理員元件能夠管理系統日誌。此物件包含下列資訊：

uploadToCloudWatch

(選用) 您可以將系統日誌上傳至 CloudWatch Logs。

預設：false

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。只有在您將 [Greengrass 核元件](#)設定為輸出 JSON 格式日誌時，才會套用此最低層級。若要啟用 JSON 格式日誌，請JSON為[記錄格式](#)參數()指定 logging.format。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) Greengrass 系統日誌檔案的總大小上限，以您在 中指定的單位表示 diskSpaceLimitUnit。在 Greengrass 系統日誌檔案的總大小超過此總大小上限之後，AWS IoT Greengrass Core 軟體會刪除最舊的 Greengrass 系統日誌檔案。

此參數等同於 [Greengrass 核心元件](#) 的 [日誌大小限制](#) 參數 (totalLogsSizeKB)。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為 Greengrass 系統日誌大小總計上限。

diskSpaceLimitUnit

(選用) 的單位 diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

deleteLogFileAfterCloudUpload

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

componentLogsConfiguration

(選用) 核心裝置上元件的日誌組態清單。此清單中的每個組態都會定義元件或應用程式的日誌組態。日誌管理員元件會將這些元件日誌上傳至 CloudWatch Logs

每個物件都包含下列資訊：

componentName

此日誌組態的元件或應用程式名稱。您可以指定 Greengrass 元件的名稱或其他值來識別此日誌群組。

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。此最低層級僅適用於此元件的日誌使用特定的 JSON 格式，您可以在 GitHub 的記錄 [AWS IoT Greengrass 模組](#) 儲存庫中找到該格式。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG

- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) 此元件的所有日誌檔案總大小上限，以您在 中指定的單位表示 diskSpaceLimitUnit。此元件日誌檔案的總大小超過此總大小上限後，AWS IoT Greengrass 核心軟體會刪除此元件最舊的日誌檔案。

此參數與 [Greengrass 核元件的日誌大小限制](#) 參數 (totalLogsSizeKB) 有關。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為此元件的總日誌大小上限。

diskSpaceLimitUnit

(選用) 的單位 diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

logFileDirectoryPath

(選用) 包含此元件日誌檔案的資料夾路徑。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

預設：*/greengrass/v2/logs*。

logFileRegex

(選用) 規則表達式，指定元件或應用程式使用的日誌檔案名稱格式。日誌管理員元件使用此規則表達式來識別 資料夾中的日誌檔案 logFileDirectoryPath。

您不需要為列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件指定此參數。

如果您的元件或應用程式輪換日誌檔案，請指定符合輪換日誌檔案名稱的 regex。例如，您可以指定 **hello_world\\\\w*.log** 上傳 Hello World 應用程式的日誌。\\\\w* 模

式符合零個或多個單字字元，其中包含英數字元和底線。此 regex 會比對名稱中含和不含時間戳記的日誌檔案。在此範例中，日誌管理員會上傳下列日誌檔案：

- hello_world.log – Hello World 應用程式的最新日誌檔案。
- hello_world_2020_12_15_17_0.log – Hello World 應用程式的較舊日誌檔案。

預設：*componentName*\\\\w*.log，其中 *componentName* 是此日誌組態的元件名稱。

deleteLogFileAfterCloudUpload

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

multiLineStartPattern

(選用) 規則表達式，可識別新行上的日誌訊息何時是新的日誌訊息。如果規則表達式不符合新行，日誌管理員元件會將新行附加到前一行的日誌訊息。

根據預設，日誌管理員元件會檢查該行是否以空格字元開頭，例如標籤或空格。如果沒有，日誌管理員會將該行視為新的日誌訊息處理。否則，它會將該行附加到目前的日誌訊息。此行為可確保日誌管理員元件不會分割跨越多行的訊息，例如堆疊追蹤。

periodicUploadIntervalSec

(選用) 以秒為單位的期間，日誌管理員元件會檢查要上傳的新日誌檔案。

預設：300(5 分鐘)

下限：0.000001(1 微秒)

Example範例：組態合併更新

下列範例組態指定 將系統日誌和com.example.HelloWorld元件日誌上傳至 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

(選用) 日誌管理員元件上傳之日誌的組態。此物件包含下列資訊：

systemLogsConfiguration

(選用) AWS IoT Greengrass 核心軟體系統日誌的組態。指定此組態，讓日誌管理員元件能夠管理系統日誌。此物件包含下列資訊：

uploadToCloudWatch

(選用) 您可以將系統日誌上傳至 CloudWatch Logs。

預設：false

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。只有在您將 [Greengrass 核元件](#) 設定為輸出 JSON 格式日誌時，才會套用此最低層級。若要啟用 JSON 格式日誌，請 JSON 為 [記錄格式](#) 參數 () 指定 logging.format。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) Greengrass 系統日誌檔案的總大小上限，以您在 中指定的單位表示 diskSpaceLimitUnit。在 Greengrass 系統日誌檔案的總大小超過此總大小上限之後，AWS IoT Greengrass Core 軟體會刪除最舊的 Greengrass 系統日誌檔案。

此參數等同於 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (totalLogsSizeKB)。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為 Greengrass 系統日誌大小總計上限。

diskSpaceLimitUnit

(選用) 的單位 diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

deleteLogFileAfterCloudUpload

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

componentLogsConfiguration

(選用) 核心裝置上元件的日誌組態清單。此清單中的每個組態都會定義元件或應用程式的日誌組態。日誌管理員元件會將這些元件日誌上傳至 CloudWatch Logs

每個物件都包含下列資訊：

componentName

此日誌組態的元件或應用程式名稱。您可以指定 Greengrass 元件的名稱或其他值來識別此日誌群組。

minimumLogLevel

(選用) 要上傳的日誌訊息最低層級。此最低層級僅適用於此元件的日誌使用特定的 JSON 格式時，您可以在 GitHub 上的 [AWS IoT Greengrass 記錄模組](#) 儲存庫中找到該格式。

從下列日誌層級中選擇，此處依層級順序列出：

- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

diskSpaceLimit

(選用) 此元件的所有日誌檔案總大小上限，以您在 中指定的單位表示diskSpaceLimitUnit。此元件日誌檔案的總大小超過此總大小上限後，AWS IoT Greengrass 核心軟體會刪除此元件最舊的日誌檔案。

此參數與 [Greengrass 核元件](#) 的 [日誌大小限制](#) 參數 (totalLogsSizeKB) 有關。Core AWS IoT Greengrass 軟體會使用兩個值的最小值作為此元件的總日誌大小上限。

diskSpaceLimitUnit

(選用) 的單位diskSpaceLimit。您可以從以下選項中選擇：

- KB – KB
- MB – MB
- GB – GB

預設：KB

logFileDirectoryPath

包含此元件日誌檔案的資料夾路徑。

若要上傳 Greengrass 元件的日誌，請指定 ***/greengrass/v2/logs***，並將 取代 ***/greengrass/v2***為您的 Greengrass 根資料夾。

logFileRegex

規則表達式，指定元件或應用程式使用的日誌檔案名稱格式。日誌管理員元件使用此規則表達式來識別 資料夾中的日誌檔案logFileDirectoryPath。

若要上傳 Greengrass 元件的日誌，請指定符合輪換日誌檔案名稱的 regex。例如，您可以指定 **com.example.HelloWorld\\w*.log**上傳 Hello World 元件的日誌。\\w* 模式符合零個或多個單字字元，其中包含英數字元和底線。此 regex 會比對名稱中含和不含時間戳記的日誌檔案。在此範例中，日誌管理員會上傳下列日誌檔案：

- `com.example.HelloWorld.log` – Hello World 元件的最新日誌檔案。
- `com.example.HelloWorld_2020_12_15_17_0.log` – Hello World 元件的較舊日誌檔案。Greengrass 核將輪換時間戳記新增至日誌檔案。

`deleteLogFileAfterCloudUpload`

(選用) 您可以在日誌管理員元件將日誌上傳至 CloudWatch Logs 後刪除日誌檔案。

預設：false

`multiLineStartPattern`

(選用) 規則表達式，可識別新行上的日誌訊息何時是新的日誌訊息。如果規則表達式不符合新行，日誌管理員元件會將新行附加到上一行的日誌訊息。

根據預設，日誌管理員元件會檢查該行是否以空格字元開頭，例如標籤或空格。如果沒有，日誌管理員會將該行視為新的日誌訊息處理。否則，它會將該行附加到目前的日誌訊息。此行為可確保日誌管理員元件不會分割跨越多行的訊息，例如堆疊追蹤。

`periodicUploadIntervalSec`

(選用) 以秒為單位的期間，日誌管理員元件會檢查要上傳的新日誌檔案。

預設：300(5 分鐘)

下限：0.000001(1 微秒)

Example範例：組態合併更新

下列範例組態指定 將系統日誌和`com.example.HelloWorld`元件日誌上傳至 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"
}
```

Usage

日誌管理員元件會上傳至下列日誌群組和日誌串流。

2.1.0 and later

日誌群組名稱

```
/aws/greengrass/componentType/region/componentName
```

日誌群組名稱使用下列變數：

- **componentType** – 元件的類型，可以是下列其中一項：
 - **GreengrassSystemComponent** – 此日誌群組包含核心和外掛程式元件的日誌，這些元件在與 Greengrass 核心相同的 JVM 中執行。元件是 [Greengrass 核](#)的一部分。
 - **UserComponent** – 此日誌群組包含一般元件、Lambda 元件和裝置上的其他應用程式的日誌。元件不是 Greengrass 核的一部分。

如需詳細資訊，請參閱[元件類型](#)。

- **region** – 核心裝置使用的 AWS 區域。
- **componentName** – 元件的名稱。對於系統日誌，此值為 System。

日誌串流名稱

```
/date/thing/thingName
```

日誌串流名稱使用下列變數：

- date – 日誌的日期，例如 2020/12/15。日誌管理員元件使用 yyyy/MM/dd 格式。
- thingName – 核心裝置的名稱。

Note

如果物件名稱包含冒號 (:)，則日誌管理員會以加號 () 取代冒號+。

2.0.x

日誌群組名稱

```
/aws/greengrass/componentType/region/componentName
```

日誌群組名稱使用下列變數：

- componentType – 元件的類型，可以是下列其中一項：
 - GreengrassSystemComponent – 元件是 [Greengrass 核](#) 的一部分。
 - UserComponent – 元件不屬於 Greengrass 核。日誌管理員會將此類型用於裝置的 Greengrass 元件和其他應用程式。
- region – 核心裝置使用的 AWS 區域。
- componentName – 元件的名稱。對於系統日誌，此值為 System。

日誌串流名稱

```
/date/deploymentTargets/thingName
```

日誌串流名稱使用下列變數：

- date – 日誌的日期，例如 2020/12/15。日誌管理員元件使用 yyyy/MM/dd 格式。
- deploymentTargets – 部署包含 元件的物件。日誌管理員元件會以斜線分隔每個目標。如果元件因為本機部署而在核心裝置上執行，則此值為 LOCAL_DEPLOYMENT。

假設您有一個名為 的核心裝置MyGreengrassCore，而核心裝置有兩個部署的範例：

- 以核心裝置為目標的部署，MyGreengrassCore。
- 以名為 的物件群組為目標的部署MyGreengrassCoreGroup，其中包含核心裝置。

deploymentTargets 此核心裝置的 為 thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup。

- thingName – 核心裝置的名稱。

日誌項目的格式。

Greengrass 核會以字串或 JSON 格式寫入日誌檔案。對於系統日誌，您可以透過設定 logging 項目 format 的欄位來控制格式。您可以在 Greengrass 核元件的組態檔案中找到 logging 項目。如需詳細資訊，請參閱 [Greengrass 核組態](#)。

文字格式為自由格式，並接受任何字串。下列機群狀態服務訊息是字串格式化記錄的範例：

```
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}
```

如果您想要使用 [Greengrass CLI 日誌](#) 命令檢視日誌或以程式設計方式與日誌互動，您應該使用 JSON 格式。下列範例概述 JSON 形狀：

```
{
  "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
}
```

若要控制元件日誌的輸出，您可以使用 minimumLogLevel 組態選項。若要使用此選項，您的元件必須以 JSON 格式寫入其日誌項目。您應該使用與系統日誌檔案相同的格式。

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.3.11	錯誤修正和改善 修正 Log Manager 執行期組態因上傳日誌檔案的過時資訊而無限期成長的問題。
2.3.10	新功能 - 新增組態金鑰 (<code>updateToTlogIntervalSec</code>)，以控制日誌上傳事件詳細資訊保留至本機交易日誌 () 的頻率 <code>config.tlog</code>。 錯誤修正和改善 - 改善日誌管理員以重新整理 cloudwatch 用戶端的通訊端連線錯誤。
2.3.9	針對 Greengrass nucleus 2.14.0 版更新版本。
2.3.8	針對 Greengrass nucleus 2.13.0 版更新版本。

版本	變更
2.3.7	針對 Greengrass nucleus 2.12.0 版更新版本。
2.3.6	錯誤修正和改善 調整特定錯誤的日誌層級。
2.3.5	改進 改善日誌上傳速度。 針對 Greengrass nucleus 2.11.0 版更新版本。
2.3.4	錯誤修正和改善 - 新增將 <code>periodicUploadIntervalSec</code> 參數設定為分數的支援。最小值為 1 微秒。 - 修正日誌管理員不遵守 CloudWatch <code>putLogEvents</code> 限制的問題。
2.3.3	針對 Greengrass nucleus 2.10.0 版更新版本。
2.3.2	錯誤修正和改善 - 改善空間管理，避免日誌檔案在上傳之前遭到刪除。 - 修正快取管理的問題。 - 其他次要錯誤修正和改善。
2.3.1	錯誤修正和改善 - 修正 以多個作用中日誌檔案為目標的檔案群組將重複項目上傳至 CloudWatch 的問題。 - 其他次要錯誤修正和改善。

版本	變更
2.3.0	 Note 當您升級至 log Manager 2.3.0 時，我們建議您升級至 Greengrass 核 2.9.1。 新功能 透過處理和直接上傳作用中日誌檔案，而不是等待輪換新檔案，以減少日誌延遲。 錯誤修正和改善 - 改善使用唯一名稱輪換檔案時的日誌輪換支援。 - 其他次要錯誤修正和改善。
2.2.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.2.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.2.6	針對 Greengrass nucleus 2.7.0 版更新版本。
2.2.5	針對 Greengrass nucleus 2.6.0 版更新版本。
2.2.4	錯誤修正和改善 - 改善處理無效組態時的穩定性。 - 其他次要修正和改進。
2.2.3	錯誤修正和改善 - 改善元件重新啟動或遇到錯誤的特定情況下的穩定性。 - 修正在某些情況下大型日誌訊息和大型日誌檔案無法上傳的問題。 - 修正此元件如何處理組態重設更新的問題。 - 修正nulldiskSpaceLimit 組態值阻止元件部署的問題。

版本	變更
2.2.2	錯誤修正和改善 新增對大於 256 KB 之日誌訊息的支援。日誌管理員元件會將這些大型日誌訊息分割成具有相同日誌事件時間戳記的多個訊息。
2.2.1	針對 Greengrass nucleus 2.5.0 版更新版本。
2.2.0	新功能 新增 <code>componentLogsConfigurationMap</code> 組態參數，以支援元件日誌組態的映射格式。映射中的每個 <code>componentName</code> 物件都會定義元件或應用程式的日誌組態。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.1	錯誤修正和改善 修正系統日誌組態在特定情況下未更新的問題。
2.1.0	錯誤修正和改善 - 使用 <code>logFileDirectoryPath</code> 和 的預設值 <code>logFileRegex</code>，適用於列印為標準輸出 (stdout) 和標準錯誤 (stderr) 的 Greengrass 元件。 - 將日誌上傳至 CloudWatch Logs 時，透過設定的網路代理正確路由流量。 - 正確處理日誌串流名稱中的冒號字元 (:)。CloudWatch Logs 日誌串流名稱不支援冒號。 - 從日誌串流中移除物件群組名稱，以簡化日誌串流名稱。 - 移除在正常行為期間列印的錯誤日誌訊息。
2.0.x	初始版本。

機器學習元件

AWS IoT Greengrass 提供下列機器學習元件，您可以將這些元件部署到支援的裝置，以使用在 Amazon SageMaker AI 中訓練的模型或儲存在 Amazon S3 中的您自己的預先訓練模型[來執行機器學習推論](#)。

AWS 提供以下類別的機器學習元件：

- 模型元件 - 包含機器學習模型做為 Greengrass 成品。
- 執行期元件 - 包含在 Greengrass 核心裝置上安裝機器學習架構及其相依性的指令碼。
- 推論元件 - 包含推論程式碼，並包含元件相依性，以安裝機器學習架構並下載預先訓練的機器學習模型。

您可以在 AWS 提供的機器學習元件中使用範例推論程式碼和預先訓練的模型，以使用 DLR 和 TensorFlow Lite 執行影像分類和物件偵測。若要使用存放在 Amazon S3 中的自有模型執行自訂機器學習推論，或使用不同的機器學習架構，您可以使用這些公有元件的配方做為範本來建立自訂機器學習元件。如需詳細資訊，請參閱[自訂您的機器學習元件](#)。

AWS IoT Greengrass 也包含 AWS 提供的元件，可在 Greengrass 核心裝置上管理 SageMaker AI Edge Manager 代理程式的安裝和生命週期。透過 SageMaker AI Edge Manager，您可以直接在核心裝置上使用 Amazon SageMaker AI Neo 編譯模型。如需詳細資訊，請參閱[在 Greengrass 核心裝置上使用 Amazon SageMaker AI Edge Manager](#)。

下表列出 中可用的機器學習元件 AWS IoT Greengrass。

Note

數個 AWS 提供的元件取決於 Greengrass 核的特定次要版本。由於此相依性，當您將 Greengrass 核更新為新的次要版本時，您需要更新這些元件。如需每個元件所依賴之核的特定版本的資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

當元件同時具有通用和 Lambda 的元件類型時，元件的目前版本是通用類型，而元件的先前版本是 Lambda 類型。

元件	說明	元件類型	支援的作業系統	開放原始碼
SageMaker AI Edge Manager	在 Greengrass 核心裝置上部署 Amazon SageMaker AI Edge Manager 代理程式。	一般	Linux、Windows	否
DLR 影像分類	使用 DLR 影像分類模型存放區和 DLR 執行期元件做為相依性的推論元件，以安裝 DLR、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。	一般	Linux、Windows	否
DLR 物件偵測	使用 DLR 物件偵測模型存放區和 DLR 執行期元件做為相依性的推論元件，以安裝 DLR、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。	一般	Linux、Windows	否

元件	說明	元件類型	支援的作業系統	開放原始碼
DLR 影像分類模型存放區	模型元件，其中包含做為 Greengrass 成品的範例 ResNet-50 影像分類模型。	一般	Linux、Windows	否
DLR 物件偵測模型存放區	模型元件，其中包含做為 Greengrass 成品的範例 YOLOv3 物件偵測模型。	一般	Linux、Windows	否
DLR 執行時間	包含安裝指令碼的執行期元件，用於在 Greengrass 核心裝置上安裝 DLR 及其相依性。	一般	Linux、Windows	否

元件	說明	元件類型	支援的作業系統	開放原始碼
TensorFlow Lite 影像分類	使用 TensorFlow Lite 影像分類模型存放區和 TensorFlow Lite 執行期元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。	一般	Linux、Windows	否
TensorFlow Lite 物件偵測	使用 TensorFlow Lite 物件偵測模型存放區和 TensorFlow Lite 執行期元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。	一般	Linux、Windows	否

元件	說明	元件類型	支援的作業系統	開放原始碼
TensorFlow Lite 影像分類模型存放區	模型元件，其中包含做為 Greengrass 成品的範例 MobileNet v1 模型。	一般	Linux、Windows	否
TensorFlow Lite 物件偵測模型存放區	模型元件，其中包含做為 Greengrass 成品的範例單一鏡頭偵測 (SSD) MobileNet 模型。	一般	Linux、Windows	否
TensorFlow Lite 執行時間	包含安裝指令碼的執行期元件，用於在 Greengrass 核心裝置上安裝 TensorFlow Lite 及其相依性。	一般	Linux、Windows	否

SageMaker AI Edge Manager

Important

SageMaker AI Edge Manager 已於 2024 年 4 月 26 日終止。如需繼續將模型部署到邊緣裝置的詳細資訊，請參閱 [SageMaker AI Edge Manager 生命週期結束](#)。

Amazon SageMaker AI Edge Manager 元件 (aws.greengrass.SageMakerEdgeManager) 安裝 SageMaker AI Edge Manager 代理程式二進位檔。

SageMaker AI Edge Manager 為邊緣裝置提供模型管理，讓您可以在邊緣裝置機群上最佳化、保護、監控和維護機器學習模型。SageMaker AI Edge Manager 元件會在您的核心裝置上安裝和管理 SageMaker AI Edge Manager 代理程式的生命週期。您也可以使用 SageMaker AI Edge Manager 封裝和使用 SageMaker AI Neo 編譯模型作為 Greengrass 核心裝置上的模型元件。如需在核心裝置上使用 SageMaker AI Edge Manager 代理程式的詳細資訊，請參閱 [在 Greengrass 核心裝置上使用 Amazon SageMaker AI Edge Manager](#)。

SageMaker AI Edge Manager 元件 v1.3.x 安裝 Edge Manager 代理程式二進位 v1.20220822.836f3023。如需 Edge Manager 代理程式二進位版本的詳細資訊，請參閱 [Edge Manager 代理程式](#)。

 Note

SageMaker AI Edge Manager 元件僅適用於下列項目 AWS 區域：

- 美國東部 (俄亥俄)
- 美國東部 (維吉尼亞北部)
- 美國西部 (奧勒岡)
- 歐洲 (法蘭克福)
- 歐洲 (愛爾蘭)
- 亞太地區 (東京)

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在 Amazon Linux 2 上執行的 Greengrass 核心裝置、以 Debian 為基礎的 Linux 平台 (x86_64 或 Armv8) 或 Windows (x86_64)。如果您沒有帳戶，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。
- [Python](#) 3.6 或更新版本，包括 pip 安裝在核心裝置上的 Python 版本。
- 使用下列項目設定的 [Greengrass 裝置角色](#)：
 - 允許 credentials.iot.amazonaws.com 和 sagemaker.amazonaws.com 擔任角色的信任關係，如下列 IAM 政策範例所示。

```
{
  "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"
}
]
```

- [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM 受管政策。
- s3:PutObject 動作，如下列 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- 在與 Greengrass 核心裝置相同的 AWS 帳戶 和 中建立 AWS 區域 的 Amazon S3 儲存貯體。SageMaker AI Edge Manager 需要 S3 儲存貯體來建立邊緣裝置機群，並儲存範例資料，避免在裝置上執行推論。如需建立 S3 儲存貯體的資訊，請參閱 [Amazon S3 入門](#)。
- 使用與 Greengrass 核心裝置相同 AWS IoT 角色別名的 SageMaker AI 邊緣裝置機群。如需詳細資訊，請參閱[建立邊緣裝置機群](#)。
- 您的 Greengrass 核心裝置已註冊為 SageMaker AI Edge 裝置機群中的邊緣裝置。邊緣裝置名稱必須與核心裝置的 AWS IoT 物件名稱相符。如需詳細資訊，請參閱[註冊您的 Greengrass 核心裝置](#)。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	連線埠	必要	說明
edge.sagemaker. <i>region</i> .amazonaws.com	443	是	檢查裝置註冊狀態，並將指標傳送至 SageMaker AI。
*.s3.amazonaws.com	443	是	將擷取資料上傳到您指定的 S3 儲存貯體。 您可以將取代*為您上傳資料的每個儲存貯體的名稱。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

1.3.5 and 1.3.6

下表列出此元件 1.3.5 和 1.3.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式
字符交換服務	>=0.0.0	硬式

1.3.4

下表列出此元件 1.3.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式
字符交換服務	>=0.0.0	硬式

1.3.3

下表列出此元件 1.3.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式
權杖交換服務	>=0.0.0	硬式

1.3.2

下表列出此元件 1.3.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式
權杖交換服務	>=0.0.0	硬式

1.3.1

下表列出此元件 1.3.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式
權杖交換服務	>=0.0.0	硬式

1.1.1 - 1.3.0

下表列出此元件 1.1.1 - 1.3.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式
字符交換服務	>=0.0.0	硬式

1.1.0

下表列出此元件 1.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式
字符交換服務	>=0.0.0	硬式

1.0.3

下表列出此元件 1.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

相依性	相容版本	相依性類型
字符交換服務	>=0.0.0	硬式

1.0.1 and 1.0.2

下表列出此元件 1.0.1 和 1.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式
權杖交換服務	>=0.0.0	硬式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式
權杖交換服務	>=0.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

Note

本節說明您在 元件中設定的組態參數。如需對應 SageMaker AI Edge Manager 組態的詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的 [Edge Manager Agent](#)。

DeviceFleetName

包含 Greengrass 核心裝置的 SageMaker AI Edge Manager 裝置機群名稱。

部署此元件時，您必須在組態更新中指定此參數的值。

BucketName

您上傳擷取推論資料的 S3 儲存貯體名稱。儲存貯體名稱必須包含字串 sagemaker。

如果您將 CaptureDataDestination 設定為 Cloud，或者將 CaptureDataPeriodicUpload 設定為 true，則您必須在部署此元件時，於組態更新中指定此參數的值。

Note

擷取資料是一種 SageMaker AI 功能，可用來將推論輸入、推論結果和其他推論資料上傳至 S3 儲存貯體或本機目錄，以供未來分析。如需搭配 SageMaker AI Edge Manager 使用擷取資料的詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的[管理模型](#)。

CaptureDataBatchSize

(選用) 代理程式處理的擷取資料請求批次大小。此值必須小於您在 中指定的緩衝區大小 CaptureDataBufferSize。建議您不要超過緩衝區大小的一半。

當緩衝區中的請求數量符合 CaptureDataBatchSize 數量，或 CaptureDataPushPeriodSeconds 間隔經過時，代理程式會處理請求批次，以先發生者為準。

預設：10

CaptureDataBufferSize

(選用) 存放在緩衝區中的擷取資料請求數目上限。

預設：30

CaptureDataDestination

(選用) 存放擷取資料的目的地。此參數可以有下列值：

- Cloud- 將擷取的資料上傳到您在 中指定的 S3 儲存貯體 BucketName。

- Disk- 將擷取的資料寫入元件的工作目錄。

如果您指定 Disk，您也可以選擇將 `CaptureDataPeriodicUpload` 設定為 `true`，定期將擷取的資料上傳到您的 S3 儲存貯體。

預設：Cloud

`CaptureDataPeriodicUpload`

(選用) 字串值，指定是否定期上傳擷取的資料。支援的值為 `true` 和 `false`。

`true` 如果您將設定為 Disk，且也希望代理程式定期上傳 S3 儲存貯體擷取的資料，請將此參數 `CaptureDataDestination` 設定為 `S3`。

預設：false

`CaptureDataPeriodicUploadPeriodSeconds`

(選用) SageMaker AI Edge Manager 代理程式將擷取資料上傳至 S3 儲存貯體的間隔，以秒為單位。如果您將 `CaptureDataPeriodicUpload` 設定為 `true`，請使用此參數。

預設：8

`CaptureDataPushPeriodSeconds`

(選用) SageMaker AI Edge Manager 代理程式處理緩衝區批次擷取資料請求的間隔，以秒為單位。

當緩衝區中的請求數量符合 `CaptureDataBatchSize` 數量，或 `CaptureDataPushPeriodSeconds` 間隔經過時，代理程式會處理請求批次，以先發生者為準。

預設：4

`CaptureDataBase64EmbedLimit`

(選用) SageMaker AI Edge Manager 代理程式上傳的擷取資料位元組大小上限。

預設：3072

`FolderPrefix`

(選用) 代理程式寫入擷取資料的資料夾名稱。如果您將 `CaptureDataDestination` 設定為 Disk，代理程式會在指定的目錄中建立資料夾 `CaptureDataDiskPath`。如果您將

CaptureDataDestination 設定為 Cloud，或者將 CaptureDataPeriodicUpload 設定為 true，則代理程式會在 S3 儲存貯體中建立 資料夾。

預設：sme-capture

CaptureDataDiskPath

此功能可在 SageMaker AI Edge Manager 元件的 v1.1.0 和更新版本中使用。

(選用) 代理程式建立擷取資料資料夾的資料夾路徑。如果您將 CaptureDataDestination 設定為 Disk，代理程式會在此目錄中建立擷取的資料資料夾。如果您未指定此值，代理程式會在元件的工作目錄中建立擷取的資料資料夾。使用 FolderPrefix 參數來指定擷取資料資料夾的名稱。

預設：`/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager/capture`

LocalDataRootPath

此功能可在 SageMaker AI Edge Manager 元件的 v1.2.0 和更新版本中使用。

(選用) 此元件在核心裝置上存放下列資料的路徑：

- 將 DbEnable 設定為 true 時，執行時間資料的本機資料庫 true。
- 當您將 DeploymentEnable 設定為 true 時，此元件會自動下載的 SageMaker AI Neo 編譯模型 true。

預設：`/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager`

DbEnable

(選用) 您可以讓此元件在本機資料庫中存放執行時間資料以保留資料，以防元件故障或裝置斷電。

此資料庫在核心裝置的檔案系統上需要 5 MB 的儲存空間。

預設：false

DeploymentEnable

此功能可在 SageMaker AI Edge Manager 元件的 v1.2.0 和更新版本中使用。

(選用) 您可以啟用此元件，從您上傳至 Amazon S3 的模型中自動擷取 SageMaker AI Neo 編譯的模型。將新模型上傳至 Amazon S3 之後，請使用 SageMaker AI Studio 或 SageMaker AI API

將新模型部署至此核心裝置。當您啟用此功能時，您可以將新模型部署到核心裝置，而不需要建立 AWS IoT Greengrass 部署。

 Important

若要使用此功能，您必須將 `DbEnable` 設定為 `true`。此功能使用本機資料庫來追蹤從擷取的模型 AWS 雲端。

預設：`false`

`DeploymentPollInterval`

此功能可在 SageMaker AI Edge Manager 元件的 v1.2.0 和更新版本中使用。

(選用) 此元件檢查要下載的新模型之間的時間量 (分鐘)。當您將 `DeploymentEnable` 設定為 `true` 時，此選項適用 `true`。

預設：1440(1 天)

`DLRBackendOptions`

此功能可在 SageMaker AI Edge Manager 元件的 v1.2.0 和更新版本中使用。

(選用) 此元件使用的 DLR 執行期中要設定的 DLR 執行期旗標。您可以設定下列旗標：

- `TVM_TENSORRT_CACHE_DIR` – 啟用 TensorRT 模型快取。指定具有讀取/寫入許可的現有資料夾的絕對路徑。
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB` – 指派 TensorRT 模型快取資料夾的上限。當目錄大小超過此限制時，最少使用的快取引擎會被刪除。預設值為 512 MB。

例如，您可以將此參數設定為下列值，以啟用 TensorRT 模型快取，並將快取大小限制為 800 MB。

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;  
TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

(選用) 指定是否啟用偵錯記錄的字串值。支援的值為 `true` 和 `false`。

預設：`false`

UnixSocketName

(選用) 核心裝置上 SageMaker AI Edge Manager 通訊端檔案描述項的位置。

預設：`/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example範例：組態合併更新

下列範例組態指定核心裝置是 *MyEdgeDeviceFleet* 的一部分，且代理程式會將擷取資料寫入裝置和 S3 儲存貯體。此組態也會啟用偵錯記錄。

```
{
  "DeviceFleetName": "MyEdgeDeviceFleet",
  "BucketName": "amzn-s3-demo-bucket",
  "CaptureDataDestination": "Disk",
  "CaptureDataPeriodicUpload": "true",
  "SagemakerEdgeLogVerbose": "true"
}
```

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C:\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.3.6	針對 Greengrass 核 2.12.5 版更新版本。
1.3.5	針對 Greengrass nucleus 2.12.0 版更新版本。
1.3.4	針對 Greengrass nucleus 2.11.0 版更新版本。
1.3.3	針對 Greengrass nucleus 2.10.0 版更新版本。
1.3.2	針對 Greengrass nucleus 2.9.0 版更新版本。
1.3.1	針對 Greengrass nucleus 2.8.0 版更新版本。
1.3.0	新功能 - 新增對 TensorRT 快取磁碟大小管理的支援。 - 將選用 TVM_TENSORRT_CACHE_DISK_SIZE_MB 旗標新增至 DLRBackendOptions 參數，以設定磁碟上快取模型的大小限制。 改善項目 - 提供改善的預測並行。這有助於更好地使用裝置加速器引擎，例如 GPUs。
1.2.0	新功能 - 新增此元件的支援，以自動擷取您上傳至 Amazon S3 的 SageMaker AI Neo 編譯模型。啟用此功能時，您可以將新模型部署到核心裝置，而不需要建立 AWS IoT Greengrass 部署。 - 新增此元件用來保留執行期資料的備份資料庫支援，以防元件故障或裝置斷電。

版本	變更
	• 新增支援，讓您在設定此元件時設定 DLR 執行期旗標。
1.1.1	針對 Greengrass nucleus 2.7.0 版更新版本。
1.1.0	新功能 • 新增對執行 Amazon Linux 2 的 Greengrass 核心裝置的支援。 • 新增新的CaptureDataDiskPath 組態參數。您可以使用此參數來指定裝置上的擷取資料資料夾路徑。 錯誤修正和改善 • 針對 Greengrass nucleus 2.5.0 版更新版本。
1.0.3	針對 Greengrass nucleus 2.4.0 版更新版本。
1.0.2	錯誤修正和改善 更新元件生命週期中的安裝指令碼。您的核心裝置現在必須在裝置上安裝 Python 3.6 或更新版本，包括pip適用於您 Python 版本的，才能部署此元件。
1.0.1	針對 Greengrass nucleus 2.3.0 版更新版本。
1.0.0	初始版本。

DLR 影像分類

DLR 影像分類元件 (aws.greengrass.DLRImageClassification) 包含使用 [Deep Learning Runtime](#) 和 resnet-50 模型執行影像分類推論的範例推論程式碼。此元件使用變體[DLR 影像分類模型存放區](#)和[DLR 執行時間](#)元件做為相依性，以下載 DLR 和範例模型。

若要搭配自訂訓練的 DLR 模型使用此推論元件，[請建立相依模型存放區元件的自訂版本](#)。若要使用您自己的自訂推論程式碼，您可以使用此元件的配方做為範本來[建立自訂推論元件](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)

- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

- 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

- 選取界面選項。
- 選取舊版攝影機以啟用舊版攝影機堆疊。
- 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.13 and 2.1.14

下表列出此元件 2.1.13 和 2.1.14 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.4 - 2.1.5

下表列出此元件 2.1.4 至 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式
DLR 影像分類模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	~2.0.0	軟式
DLR 影像分類模型存放區	~2.0.0	硬式
DLR	~1.3.0	軟式

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.1.x

accessControl

(選用) 包含[授權政策](#)的物件，允許元件將訊息發佈至預設通知主題。

預設：

```
{
  "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

(選用) 您要發佈推論結果的主題。如果您修改此值，則也必須修改 accessControl 參數resources中的 值，以符合您的自訂主題名稱。

預設：ml/dlr/image-classification

Accelerator

您要使用的加速器。支援的值為 cpu 和 gpu。

相依模型元件中的範例模型僅支援 CPU 加速。若要搭配不同的自訂模型使用 GPU 加速，[請建立自訂模型元件](#)以覆寫公有模型元件。

預設：cpu

ImageDirectory

(選用) 推論元件讀取映像之裝置上的資料夾路徑。您可以將此值修改為您裝置上具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

 Note

如果您將 `UseCamera` 的值設定為 `true`，則會忽略此組態參數。

ImageName

(選用) 推論元件用來輸入進行預測的影像名稱。元件會在 `ImageDirectory` 中指定的資料夾中尋找映像。根據預設，元件會使用預設映像目錄中的範例映像。AWS IoT Greengrass 支援下列映像格式：`jpeg`、`png`、`jpg`和 `npv`。

預設：`cat.jpeg`

 Note

如果您將 `UseCamera` 的值設定為 `true`，則會忽略此組態參數。

InferenceInterval

(選用) 推論程式碼所做的每個預測之間的秒數。範例推論程式碼會無限期執行，並在指定的時間間隔重複其預測。例如，如果您想要使用攝影機拍攝的影像進行即時預測，您可以將此值變更為較短的間隔。

預設：`3600`

ModelResourceKey

(選用) 用於相依公有模型元件的模型。只有在您使用自訂元件覆寫公有模型元件時，才修改此參數。

預設：

```
{
  "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

(選用) 字串值，定義是否使用連接至 Greengrass 核心裝置的攝影機的影像。支援的值為 true 和 false。

當您將此值設定為 true，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像上本機執行推論。會忽略 ImageName 和 ImageDirectory 參數的值。請確定執行此元件的使用者具有攝影機存放擷取影像之位置的讀取/寫入存取權。

預設：false

Note

當您檢視此元件的配方時，UseCamera 組態參數不會出現在預設組態中。不過，您可以在部署元件時修改[組態合併更新](#)中此參數的值。

將 UseCamera 設定為 true，您還必須建立符號連結，讓推論元件從執行期元件建立的虛擬環境中存取您的攝影機。如需搭配範例推論元件使用攝影機的詳細資訊，請參閱[更新元件組態](#)。

2.0.x

MLRootPath

(選用) Linux 核心裝置上的資料夾路徑，其中推論元件會讀取映像和寫入推論結果。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權的任何位置。

預設：/greengrass/v2/work/variant.DLR/greengrass_ml

預設：/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml

Accelerator

您要使用的加速器。支援的值為 cpu 和 gpu。

相依模型元件中的範例模型僅支援 CPU 加速。若要搭配不同的自訂模型使用 GPU 加速，[請建立自訂模型元件](#)以覆寫公有模型元件。

預設：cpu

ImageName

(選用) 推論元件用來輸入進行預測的影像名稱。元件會在 中指定的資料夾中尋找映像 ImageDirectory。預設位置為 *MLRootPath*/images。AWS IoT Greengrass 支援下列影像格式：jpeg、png、jpg 和 npy。

預設：cat.jpeg

InferenceInterval

(選用) 推論程式碼所做的每個預測之間的秒數。範例推論程式碼會無限期執行，並在指定的時間間隔重複其預測。例如，如果您想要使用攝影機拍攝的影像進行即時預測，您可以將此值變更為較短的間隔。

預設：3600

ModelResourceKey

(選用) 用於相依公有模型元件的模型。只有在您使用自訂元件覆寫公有模型元件時，才修改此參數。

預設：

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRIImageClassification.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRIImageClassification.log -  
Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.14	針對 Greengrass 核 2.12.5 版更新版本。
2.1.13	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.12	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.11	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.5	所有中發行的元件 AWS 區域。

版本	變更
2.1.4	針對 Greengrass nucleus 2.4.0 版更新版本。 此版本不適用於歐洲（倫敦）(eu-west-2)。
2.1.3	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.1	新功能 • 使用深度學習執行期 v1.6.0。 • 新增對 Armv8 (AArch64) 平台上範例映像分類的支援。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。 • 為範例推論啟用攝影機整合。使用新的UseCamera 組態參數，讓範例推論程式碼存取 Greengrass 核心裝置上的攝影機，並在擷取的影像上本機執行推論。 • 新增將推論結果發佈至 的支援 AWS 雲端。使用新的PublishResultsOnTopic 組態參數來指定您要發佈結果的主題。 • 新增新的ImageDirectory 組態參數，可讓您為要執行推論的映像指定自訂目錄。 錯誤修正和改善 • 將推論結果寫入元件日誌檔案，而不是單獨的推論檔案。 • 使用 AWS IoT Greengrass 核心軟體記錄模組來記錄元件輸出。 • 使用 AWS IoT Device SDK 讀取元件組態並套用組態變更。
2.0.4	初始版本。

DLR 物件偵測

DLR 物件偵測元件 (aws.greengrass.DLRObjectDetection) 包含範例推論程式碼，以使用 [Deep Learning Runtime](#) 和範例預先訓練模型執行物件偵測推論。此元件使用變體[DLR 物件偵測模型存放區](#)和[DLR 執行時間](#)元件做為相依性，以下載 DLR 和範例模型。

若要搭配自訂訓練的 DLR 模型使用此推論元件，[請建立相依模型存放區元件的自訂版本](#)。若要使用您自己的自訂推論程式碼，您可以使用此元件的配方做為範本來[建立自訂推論元件](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
- 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

2.1.13 and 2.1.14

下表列出此元件 2.1.13 和 2.1.14 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.4 - 2.1.5

下表列出此元件 2.1.4 至 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式
DLR 物件偵測模型存放區	~2.1.0	硬式
DLR	~1.6.0	硬式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	~2.0.0	軟式
DLR 物件偵測模型存放區	~2.0.0	硬式
DLR	~1.3.0	軟式

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.1.x

accessControl

(選用) 包含[授權政策](#)的物件，允許元件將訊息發佈至預設通知主題。

預設：

```
{
  "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

(選用) 您要發佈推論結果的主題。如果您修改此值，則也必須修改 `accessControl` 參數 `resources` 中的 值，以符合您的自訂主題名稱。

預設：ml/dlr/object-detection

Accelerator

您要使用的加速器。支援的值為 `cpu` 和 `gpu`。

相依模型元件中的範例模型僅支援 CPU 加速。若要搭配不同的自訂模型使用 GPU 加速，[請建立自訂模型元件](#)以覆寫公有模型元件。

預設：`cpu`

ImageDirectory

(選用) 推論元件讀取映像之裝置上的資料夾路徑。您可以將此值修改為您裝置上具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/packages/artifacts-unarchived/component-name/object_detection/sample_images/`

Note

如果您將 的值設定為 `UseCamera true`，則會忽略此組態參數。

ImageName

(選用) 推論元件用來輸入進行預測的影像名稱。元件會在 中指定的資料夾中尋找映像 `ImageDirectory`。根據預設，元件會使用預設映像目錄中的範例映像。AWS IoT Greengrass 支援下列映像格式：`jpeg`、`png`、`jpg`和 `npz`。

預設：`objects.jpg`

Note

如果您將 的值設定為 `UseCamera true`，則會忽略此組態參數。

InferenceInterval

(選用) 推論程式碼所做的每個預測之間的秒數。範例推論程式碼會無限期執行，並在指定的時間間隔重複其預測。例如，如果您想要使用攝影機拍攝的影像進行即時預測，您可以將此值變更為較短的間隔。

預設：`3600`

ModelResourceKey

(選用) 用於相依公有模型元件的模型。只有在您使用自訂元件覆寫公有模型元件時，才修改此參數。

預設：

```
{
  "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

(選用) 字串值，定義是否使用連接至 Greengrass 核心裝置的攝影機的影像。支援的值為 true 和 false。

當您將此值設定為 true，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像上本機執行推論。會忽略 ImageName 和 ImageDirectory 參數的值。請確定執行此元件的使用者具有攝影機存放擷取影像之位置的讀取/寫入存取權。

預設：false

Note

當您檢視此元件的配方時，UseCamera 組態參數不會出現在預設組態中。不過，您可以在部署元件時修改[組態合併更新](#)中此參數的值。

將 UseCamera 設定為 true，您還必須建立符號連結，讓推論元件從執行期元件建立的虛擬環境中存取您的攝影機。如需搭配範例推論元件使用攝影機的詳細資訊，請參閱[更新元件組態](#)。

2.0.x

MLRootPath

(選用) Linux 核心裝置上的資料夾路徑，其中推論元件會讀取映像和寫入推論結果。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/work/variant.DLR/greengrass_ml`

預設：`/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

Accelerator

請勿修改。目前，加速器唯一支援的值是 `cpu`，因為相依模型元件中的模型只會針對 CPU 加速器編譯。

ImageName

(選用) 推論元件用來輸入進行預測的影像名稱。元件會在 中指定的資料夾中尋找映像 `ImageDirectory`。預設位置為 `MLRootPath/images`。AWS IoT Greengrass 支援下列影像格式：`jpeg`、`png`、`jpg` 和 `npv`。

預設：`objects.jpg`

InferenceInterval

(選用) 推論程式碼所做的每個預測之間的秒數。範例推論程式碼會無限期執行，並在指定的時間間隔重複其預測。例如，如果您想要使用攝影機拍攝的影像進行即時預測，您可以將此值變更為較短的間隔。

預設：`3600`

ModelResourceKey

(選用) 用於相依公有模型元件的模型。只有在您使用自訂元件覆寫公有模型元件時，才修改此參數。

預設：

```
{
  armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
  x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.14	針對 Greengrass 核 2.12.5 版更新版本。
2.1.13	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.12	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.11	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.6.0 版更新版本。

版本	變更
2.1.6	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.5	所有中發行的元件 AWS 區域。
2.1.4	針對 Greengrass nucleus 2.4.0 版更新版本。 此版本不適用於歐洲（倫敦）(eu-west-2)。
2.1.3	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.2	錯誤修正和改善 修正導致範例 DLR 物件偵測推論結果中有不正確週框方塊的影像擴展問題。
2.1.1	新功能 - 使用深度學習執行期 v1.6.0。 - 新增對 Armv8 (AArch64) 平台上範例物件偵測的支援。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。 - 為範例推論啟用攝影機整合。使用新的 UseCamera 組態參數，讓範例推論程式碼存取 Greengrass 核心裝置上的攝影機，並在擷取的影像上本機執行推論。 - 新增將推論結果發佈至的支援 AWS 雲端。使用新的 PublishResultsOnTopic 組態參數來指定您要發佈結果的主題。 - 新增新的 ImageDirectory 組態參數，可讓您為要在其中執行推論的映像指定自訂目錄。 錯誤修正和改善 - 將推論結果寫入元件日誌檔案，而非單獨的推論檔案。 - 使用 AWS IoT Greengrass 核心軟體記錄模組來記錄元件輸出。 - 使用 AWS IoT Device SDK 讀取元件組態並套用組態變更。
2.0.4	初始版本。

DLR 影像分類模型存放區

DLR 影像分類模型存放區是一種機器學習模型元件，其中包含預先訓練的 ResNet-50 模型做為 Greengrass 成品。此元件中使用的預先訓練模型是從 [GluonCV 模型動物園](#) 擷取，並使用 SageMaker AI Neo [深度學習執行期](#) 編譯。

[DLR 影像分類](#) 推論元件使用此元件做為模型來源的相依性。若要使用自訂訓練的 DLR 模型，[請建立此模型元件的自訂版本](#)，並將您的自訂模型包含為元件成品。您可以使用此元件的配方做為範本，來建立自訂模型元件。

Note

DLR 影像分類模型存放區元件的名稱會因版本而異。2.1.x 版和更新版本的元件名稱為 `variant.DLR.ImageClassification.ModelStore2.0.x` 版的元件名稱為 `variant.ImageClassification.ModelStore`。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x (`variant.DLR.ImageClassification.ModelStore`)
- 2.0.x (`variant.ImageClassification.ModelStore`)

類型

此元件是一般元件 (`aws.greengrass.generic`)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.12 - 2.1.14

下表列出此元件 2.1.12 版和 2.1.13 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	~2.0.0	軟式

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件不會輸出日誌。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.13	針對 Greengrass 核 2.12.5 版更新版本。
2.1.12	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.11	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.5	新功能 • 新增 Windows 核心裝置的範例映像分類模型。 • 針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.4.0 版更新版本。

版本	變更
2.1.3	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.1	新功能 新增 Armv8 (AArch64) 平台的範例 ResNet-50 影像分類模型。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。
2.0.4	初始版本。

DLR 物件偵測模型存放區

DLR 物件偵測模型存放區是一種機器學習模型元件，其中包含預先訓練的 YOLOv3 模型做為 Greengrass 成品。此元件中使用的範例模型是從 [GluonCV 模型動物園](#) 擷取，並使用 SageMaker AI Neo [深度學習執行期](#) 編譯。

[DLR 物件偵測](#) 推論元件使用此元件做為模型來源的相依性。若要使用自訂訓練的 DLR 模型，[請建立此模型元件的自訂版本](#)，並將您的自訂模型包含為元件成品。您可以使用此元件的配方做為範本，來建立自訂模型元件。

Note

DLR 物件偵測模型存放區元件的名稱因版本而異。2.1.x 版和更新版本的元件名稱為 `variant.DLR.ObjectDetection.ModelStore2.0.x` 版的元件名稱為 `variant.ObjectDetection.ModelStore`。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)

- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。


```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

- 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

- 選取界面選項。
- 選取舊版攝影機以啟用舊版攝影機堆疊。
- 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

2.1.13 and 2.1.14

下表列出此元件 2.1.13 和 2.1.14 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.1.12

下表列出此元件 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.1.5 and 2.1.6

下表列出此元件 2.1.5 和 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	~2.0.0	軟式

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件不會輸出日誌。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.14	針對 Greengrass nucleus 2.12.5 版本更新版本。
2.1.13	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.12	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.11	針對 Greengrass nucleus 2.10.0 版更新版本。

版本	變更
2.1.10	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.6	新增 CPU 模型以修正 Armv8 (AArch64) 裝置的問題。
2.1.5	新功能 • 新增 Windows 核心裝置的範例物件偵測模型。 錯誤修正和改善 • 針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.1	新功能 • 新增 Armv8 (AArch64) 平台的範例 YOLOv3 物件偵測模型。AArch64 這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。
2.0.4	初始版本。

DLR 執行時間

DLR 執行期元件 (variant.DLR) 包含指令碼，可在裝置上的虛擬環境中安裝[深度學習執行期](#) (DLR) 及其相依性。[DLR 影像分類](#) 和 [DLR 物件偵測](#) 元件使用此元件做為安裝 DLR 的相依性。元件版本 1.6.x 安裝 DLR v1.6.0，元件版本 1.3.x 安裝 DLR v1.3.0。

若要使用不同的執行時間，您可以使用此元件的配方做為範本來[建立自訂機器學習元件](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [用途](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 1.6.x
- 1.3.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。

- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

端點和連接埠

根據預設，此元件會使用安裝程式指令碼，根據核心裝置使用的平台，使用 apt、brew、yum 和 pip 命令來安裝套件。此元件必須能夠對各種套件索引和儲存庫執行傳出請求，才能執行安裝程式指令碼。若要允許此元件透過代理或防火牆的傳出流量，您必須識別核心裝置連線安裝之套件索引和儲存庫的端點。

當您識別此元件安裝指令碼所需的端點時，請考慮下列事項：

- 端點取決於核心裝置的平台。例如，執行 Ubuntu 的核心裝置會使用 apt 而非 yum 或 brew。此外，使用相同套件索引的裝置可能會有不同的來源清單，因此可能會從不同的儲存庫擷取套件。
- 使用相同套件索引的多個裝置之間的端點可能不同，因為每個裝置都有自己的來源清單，可定義擷取套件的位置。

- 端點可能會隨著時間而變更。每個套件索引都會提供您下載套件之儲存庫URLs，而套件的擁有者可以變更套件索引提供的URLs。

如需此元件安裝的相依性，以及如何停用安裝程式指令碼的詳細資訊，請參閱 [UseInstaller](#) 組態參數。

如需基本操作所需的端點和連接埠的詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

1.6.11 - 1.6.16

下表列出此元件 1.6.11 至 1.6.16 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式

1.6.10

下表列出此元件 1.6.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

1.6.9

下表列出此元件 1.6.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

1.6.8

下表列出此元件 1.6.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

1.6.6 and 1.6.7

下表列出此元件 1.6.6 和 1.6.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

1.6.4 and 1.6.5

下表列出此元件 1.6.4 和 1.6.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

1.6.3

下表列出此元件 1.6.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

1.6.2

下表列出此元件 1.6.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

1.6.1

下表列出此元件 1.6.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

1.3.x

下表列出此元件 1.3.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	~2.0.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

MLRootPath

(選用) Linux 核心裝置上的資料夾路徑，其中推論元件會讀取映像和寫入推論結果。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/work/variant.DLR/greengrass_ml`

WindowsMLRootPath

此功能適用於此元件的 v1.6.6 和更新版本。

(選用) Windows 核心裝置上的資料夾路徑，其中推論元件會讀取映像和寫入推論結果。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權的任何位置。

預設：`C:\greengrass\v2\work\variant.DLR\greengrass_ml`

UseInstaller

(選用) 字串值，定義是否使用此元件中的安裝程式指令碼來安裝 DLR 及其相依性。支援的值為 `true` 和 `false`。

`false` 如果您想要使用自訂指令碼進行 DLR 安裝，或想要在預先建置的 Linux 映像中包含執行時間相依性，請將此值設定為 `true`。若要將此元件與 AWS 提供的 DLR 推論元件搭配使用，請安裝下列程式庫，包括任何相依性，並將其提供給 `ggc_user` 執行 ML 元件的系統使用者，例如。

- [Python](#) 3.7 或更新版本，包括 `pip` 適用於您 Python 版本的。
- [深度學習執行期](#) v1.6.0
- [NumPy](#)。
- [OpenCV-Python](#)。
- 適用於 [AWS IoT Device SDK Python 的 v2](#)。
- [AWS 一般執行期 \(CRT\) Python](#)。
- [Picamera](#) (僅適用於 Raspberry Pi 裝置)。
- [awscam 模組](#) (適用於 AWS DeepLens 裝置)。
- `libGL` (適用於 Linux 裝置)

預設：`true`

用途

使用此元件，並將 `UseInstaller` 組態參數設為 `true`，以在您的裝置上安裝 DLR 及其相依性。元件會在您的裝置上設定虛擬環境，其中包含 DLR 所需的 `OpenCV` 和 `NumPy` 程式庫。

Note

此元件中的安裝程式指令碼也會安裝最新版本的其他系統程式庫，這些程式庫是在您的裝置上設定虛擬環境並使用已安裝的機器學習架構所必需。這可能會升級裝置上的現有系統程式庫。檢閱下表，以取得此元件為每個支援的作業系統安裝的程式庫清單。如果您想要自訂此安裝程序，請將 `UseInstaller` 組態參數設定為 `false`，並開發您自己的安裝程式指令碼。

平台	裝置系統上安裝的程式庫	安裝在虛擬環境中的程式庫
Armv7l	build-essential , cmake, ca-certificates , git	setuptools , wheel
Amazon Linux 2	mesa-libGL	無
Ubuntu	wget	無

當您部署推論元件時，此執行期元件會先驗證您的裝置是否已安裝 DLR 及其相依性，如果沒有，則會為您安裝它們。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/variant.DLR.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.6.16	針對 Greengrass 核更新版本 2.12.5。
1.6.12	錯誤修正和改善 修正 Windows 作業系統使用者的安裝指令碼。
1.6.11	針對 Greengrass nucleus 2.9.0 版更新版本。
1.6.10	針對 Greengrass nucleus 2.8.0 版更新版本。
1.6.9	針對 Greengrass nucleus 2.7.0 版更新版本。
1.6.8	針對 Greengrass nucleus 2.6.0 版更新版本。
1.6.7	錯誤修正和改善 - 更新UseInstaller 安裝指令碼以安裝 libGL，該 libGL 預設無法在某些 Linux 平台上使用。 - 更新UseInstaller 安裝指令碼，以一律在此元件的虛擬環境中使用 Python 3.9。此變更有助於確保與其他程式庫的相容性。
1.6.6	新功能 - 新增對執行 Windows 之核心裝置的支援。 - 新增可用於在 Windows 核心裝置上設定推論結果資料夾的新WindowsML RootPath 組態參數。
1.6.5	新功能 新增可用於停用此元件中安裝指令碼的新UseInstaller 組態參數。
1.6.4	針對 Greengrass nucleus 2.4.0 版更新版本。
1.6.3	針對 Greengrass nucleus 2.3.0 版更新版本。
1.6.2	針對 Greengrass nucleus 2.2.0 版更新版本。

版本	變更
1.6.1	新功能 - 安裝 Deep Learning Runtime v1.6.0 及其相依性。 - 新增在 Armv8 (AArch64) 平台上安裝 DLR 的支援。這擴展了對執行 NVIDIA Jetson 之 Greengrass 核心裝置的機器學習支援，例如 Jetson Nano。 錯誤修正和改善 - AWS IoT Device SDK 在虛擬環境中安裝，以讀取元件組態並套用組態變更。 - 其他次要錯誤修正和改進。
1.3.2	初始版本。安裝 DLR v1.3.0。

TensorFlow Lite 影像分類

TensorFlow Lite 影像分類元件 (`aws.greengrass.TensorFlowLiteImageClassification`) 包含使用 [TensorFlow Lite](#) 執行時間執行影像分類推論的範例推論程式碼，以及預先訓練的範例 MobileNet 1.0 量化模型。此元件使用變體 [TensorFlow Lite 影像分類模型存放區](#) 和 [TensorFlow Lite 執行時間](#) 元件做為相依性，以下載 TensorFlow Lite 執行時間和範例模型。

若要搭配自訂訓練的 TensorFlow Lite 模型使用此推論元件，[請建立相依模型存放區元件的自訂版本](#)。若要使用您自己的自訂推論程式碼，您可以使用此元件的配方做為範本來[建立自訂推論元件](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.11 and 2.1.12

下表列出此元件 2.1.11 版和 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

相依性	相容版本	相依性類型
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

accessControl

(選用) 包含[授權政策](#)的物件，允許元件將訊息發佈至預設通知主題。

預設：

```
{
  "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

(選用) 您要發佈推論結果的主題。如果您修改此值，則也必須修改 `accessControl` 參數 `resources` 中的 值，以符合您的自訂主題名稱。

預設：ml/tflite/image-classification

Accelerator

您要使用的加速器。支援的值為 `cpu` 和 `gpu`。

相依模型元件中的範例模型僅支援 CPU 加速。若要搭配不同的自訂模型使用 GPU 加速，[請建立自訂模型元件](#)以覆寫公有模型元件。

預設：cpu

ImageDirectory

(選用) 推論元件讀取映像之裝置上的資料夾路徑。您可以將此值修改為您裝置上具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

Note

如果您將 的值設定為 `UseCamera true`，則會忽略此組態參數。

ImageName

(選用) 推論元件用來輸入進行預測的影像名稱。元件會在 中指定的資料夾中尋找映像 `ImageDirectory`。根據預設，元件會使用預設映像目錄中的範例映像。AWS IoT Greengrass 支援下列映像格式：jpeg、png、jpg 和 npy。

預設：cat.jpeg

 Note

如果您將 `image` 的值設定為 `UseCamera true`，則會忽略此組態參數。

InferenceInterval

(選用) 推論程式碼所做的每個預測之間的秒數。範例推論程式碼會無限期執行，並在指定的時間間隔重複其預測。例如，如果您想要使用攝影機拍攝的影像進行即時預測，您可以將此值變更為較短的間隔。

預設：3600

ModelResourceKey

(選用) 用於相依公有模型元件的模型。只有在您使用自訂元件覆寫公有模型元件時，才修改此參數。

預設：

```
{
  "model": "TensorFlowLite-Mobilenet"
}
```

UseCamera

(選用) 字串值，定義是否使用連接至 Greengrass 核心裝置的攝影機的影像。支援的值為 `true` 和 `false`。

當您將此值設定為 `true` 時，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像上本機執行推論。會忽略 `ImageName` 和 `ImageDirectory` 參數的值。請確定執行此元件的使用者具有攝影機存放擷取影像之位置的讀取/寫入存取權。

預設：false

 Note

當您檢視此元件的配方時，`UseCamera` 組態參數不會出現在預設組態中。不過，您可以在部署元件時修改 [組態合併更新](#) 中此參數的值。

將 `UseCamera` 設定為 `true`，您還必須建立符號連結，讓推論元件從執行期元件建立的虛擬環境中存取您的攝影機。如需搭配範例推論元件使用攝影機的詳細資訊，請參閱 [更新元件組態](#)。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.12	針對 Greengrass 核 2.12.5 版更新版本。
2.1.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	初始版本。

TensorFlow Lite 物件偵測

TensorFlow Lite 物件偵測元件 (`aws.greengrass.TensorFlowLiteObjectDetection`) 包含使用 [TensorFlow Lite](#) 執行物件偵測推論的範例推論程式碼，以及預先訓練的範例單一鏡頭偵測 (SSD) MobileNet 1.0 模型。此元件使用 變體[TensorFlow Lite 物件偵測模型存放區](#)和 [TensorFlow Lite 執行時間](#)元件做為相依項，以下載 TensorFlow Lite 和範例模型。

若要搭配自訂訓練的 TensorFlow Lite 模型使用此推論元件，您可以[建立相依模型存放區元件的自訂版本](#)。若要使用您自己的自訂推論程式碼，請使用此元件的配方做為範本來[建立自訂推論元件](#)。

主題

- [版本](#)

- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
- 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

- 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

- 選取界面選項。
- 選取舊版攝影機以啟用舊版攝影機堆疊。
- 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.11 and 2.1.12

下表列出此元件 2.1.11 版和 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式
TensorFlow Lite 影像分類模型存放區	>=2.1.0 <2.2.0	硬式
TensorFlow Lite	>=2.5.0 <2.6.0	硬式

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

accessControl

(選用) 包含[授權政策](#)的物件，允許元件將訊息發佈至預設通知主題。

預設：

```
{
  "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

(選用) 您要發佈推論結果的主題。如果您修改此值，則也必須修改 `accessControl` 參數 `resources` 中的 值，以符合您的自訂主題名稱。

預設：ml/tflite/object-detection

Accelerator

您要使用的加速器。支援的值為 `cpu` 和 `gpu`。

相依模型元件中的範例模型僅支援 CPU 加速。若要搭配不同的自訂模型使用 GPU 加速，[請建立自訂模型元件](#)以覆寫公有模型元件。

預設：cpu

ImageDirectory

(選用) 推論元件讀取映像之裝置上的資料夾路徑。您可以將此值修改為您裝置上具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/packages/artifacts-unarchived/component-name/object_detection/sample_images/`

Note

如果您將 的值設定為 `UseCamera true`，則會忽略此組態參數。

ImageName

(選用) 推論元件用來輸入進行預測的影像名稱。元件會在 中指定的資料夾中尋找映像 ImageDirectory。根據預設，元件會使用預設映像目錄中的範例映像。AWS IoT Greengrass 支援下列映像格式：jpeg、png、jpg 和 npy。

預設：objects.jpg

Note

如果您將 的值設定為 UseCamera true，則會忽略此組態參數。

InferenceInterval

(選用) 推論程式碼所做的每個預測之間的秒數。範例推論程式碼會無限期執行，並在指定的時間間隔重複其預測。例如，如果您想要使用攝影機拍攝的影像進行即時預測，您可以將此值變更為較短的間隔。

預設：3600

ModelResourceKey

(選用) 用於相依公有模型元件的模型。只有在您使用自訂元件覆寫公有模型元件時，才修改此參數。

預設：

```
{
  "model": "TensorFlowLite-SSD"
}
```

UseCamera

(選用) 字串值，定義是否使用連接至 Greengrass 核心裝置的攝影機的影像。支援的值為 true 和 false。

當您將此值設定為 true 時，範例推論程式碼會存取裝置上的攝影機，並在擷取的影像上本機執行推論。會忽略 ImageName 和 ImageDirectory 參數的值。確定執行此元件的使用者具有攝影機存放擷取影像之位置的讀取/寫入存取權。

預設：false

Note

當您檢視此元件的配方時，UseCamera組態參數不會出現在預設組態中。不過，您可以在部署元件時修改[組態合併更新](#)中此參數的值。

將 UseCamera設定為 true，您還必須建立符號連結，讓推論元件從執行期元件建立的虛擬環境中存取您的攝影機。如需搭配範例推論元件使用攝影機的詳細資訊，請參閱[更新元件組態](#)。

Note

當您檢視此元件的配方時，UseCamera組態參數不會出現在預設組態中。不過，您可以在部署元件時修改[組態合併更新](#)中此參數的值。

將 UseCamera設定為 true，您還必須建立符號連結，讓推論元件從執行期元件建立的虛擬環境中存取您的攝影機。如需搭配範例推論元件使用攝影機的詳細資訊，請參閱[更新元件組態](#)。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.12	針對 Greengrass nucleus 2.12.5 版本更新版本。
2.1.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。

版本	變更
2.1.1	錯誤修正和改善 修正導致範例 TensorFlow Lite 物件偵測推論結果中有不正確週框方塊的影像擴展問題。
2.1.0	初始版本。

TensorFlow Lite 影像分類模型存放區

TensorFlow Lite 影像分類模型存放區

(`variant.TensorFlowLite.ImageClassification.ModelStore`) 是一種機器學習模型元件，其中包含預先訓練的 MobileNet v1 模型做為 Greengrass 成品。此元件中使用的範例模型是從 [TensorFlow Hub](#) 擷取，並使用 [TensorFlow Lite](#) 實作。

[TensorFlow Lite 影像分類](#) 推論元件使用此元件做為模型來源的相依性。若要使用自訂訓練的 TensorFlow Lite 模型，[請建立此模型元件的自訂版本](#)，並將自訂模型包含為元件成品。您可以使用此元件的配方做為範本，來建立自訂模型元件。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。

3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.11 and 2.1.12

下表列出此元件 2.1.11 版和 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件不會輸出日誌。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.12	針對 Greengrass nucleus 2.12.5 版本更新版本。
2.1.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	初始版本。

TensorFlow Lite 物件偵測模型存放區

TensorFlow Lite 物件偵測模型存放區

(`variant.TensorFlowLite.ObjectDetection.ModelStore`) 是一種機器學習模型元件，其中

包含預先訓練的單一鏡頭偵測 (SSD) MobileNet 模型作為 Greengrass 成品。此元件中使用的範例模型是從 [TensorFlow Hub](#) 擷取，並使用 [TensorFlow Lite](#) 實作。

[TensorFlow Lite 物件偵測](#) 推論元件使用此元件做為模型來源的相依性。若要使用自訂訓練的 TensorFlow Lite 模型，[請建立此模型元件的自訂版本](#)，並將自訂模型包含為元件成品。您可以使用此元件的配方做為範本，來建立自訂模型元件。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

- 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

- 選取界面選項。
- 選取舊版攝影機以啟用舊版攝影機堆疊。
- 重新啟動 Raspberry Pi。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.11 and 2.1.12

下表列出此元件 2.1.11 和 2.1.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件不會輸出日誌。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.12	針對 Greengrass nucleus 2.12.5 版本更新版本。
2.1.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.5.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.1.0	初始版本。

TensorFlow Lite 執行時間

TensorFlow Lite 執行期元件 (variant.TensorFlowLite) 包含指令碼，可在您裝置上的虛擬環境中安裝 [TensorFlow Lite](#) 2.5.0 版及其相依性。[TensorFlow Lite 影像分類](#)和 [TensorFlow Lite 物件偵測](#)元件使用此執行期元件做為安裝 TensorFlow Lite 的相依性。

Note

TensorFlow Lite 執行期元件 v2.5.6 和更新版本會重新安裝 TensorFlow Lite 執行期及其相依性的現有安裝。此重新安裝有助於確保核心裝置執行相容版本的 TensorFlow Lite 及其相依性。

若要使用不同的執行時間，您可以使用此元件的配方做為範本來[建立自訂機器學習元件](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [用途](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.5.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。

- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

端點和連接埠

根據預設，此元件會使用安裝程式指令碼，根據核心裝置使用的平台，使用 apt、brew、yum 和 pip 命令來安裝套件。此元件必須能夠對各種套件索引和儲存庫執行傳出請求，才能執行安裝程式指令碼。若要允許此元件透過代理或防火牆的傳出流量，您必須識別核心裝置連線安裝之套件索引和儲存庫的端點。

當您識別此元件安裝指令碼所需的端點時，請考慮下列事項：

- 端點取決於核心裝置的平台。例如，執行 Ubuntu 的核心裝置會使用 apt 而非 yum 或 brew。此外，使用相同套件索引的裝置可能會有不同的來源清單，因此可能會從不同的儲存庫擷取套件。
- 使用相同套件索引的多個裝置之間的端點可能不同，因為每個裝置都有自己的來源清單，可定義擷取套件的位置。

- 端點可能會隨著時間而變更。每個套件索引都會提供您下載套件之儲存庫URLs，而套件的擁有者可以變更套件索引提供的 URLs。

如需此元件安裝的相依性，以及如何停用安裝程式指令碼的詳細資訊，請參閱 [UseInstaller](#) 組態參數。

如需基本操作所需的端點和連接埠的詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.5.14 and 2.5.15

下表列出此元件 2.5.14 版和 2.5.15 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

2.5.13

下表列出此元件 2.5.13 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

2.5.12

下表列出此元件 2.5.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

2.5.11

下表列出此元件 2.5.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

2.5.10

下表列出此元件 2.5.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

2.5.9

下表列出此元件 2.5.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

2.5.8

下表列出此元件 2.5.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

2.5.5 - 2.5.7

下表列出此元件 2.5.5 到 2.5.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

2.5.3 and 2.5.4

下表列出此元件 2.5.3 和 2.5.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.5.2

下表列出此元件 2.5.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.5.1

下表列出此元件 2.5.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.5.0

下表列出此元件 2.5.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

MLRootPath

(選用) Linux 核心裝置上的資料夾路徑，其中推論元件會讀取映像和寫入推論結果。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權的任何位置。

預設：`/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

WindowsMLRootPath

此功能適用於此元件的 v1.6.6 和更新版本。

(選用) Windows 核心裝置上的資料夾路徑，其中推論元件會讀取映像和寫入推論結果。您可以將此值修改為裝置上執行此元件的使用者具有讀取/寫入存取權的任何位置。

預設：`C:\greengrass\v2\work\variant.DLR\greengrass_ml`

UseInstaller

(選用) 字串值，定義是否使用此元件中的安裝程式指令碼來安裝 TensorFlow Lite 及其相依性。支援的值為 true 和 false。

false 如果您想要針對 TensorFlow Lite 安裝使用自訂指令碼，或想要在預先建置的 Linux 映像中包含執行時間相依性，請將此值設定為。若要將此元件與 AWS 提供的 TensorFlow Lite 推論元件搭配使用，請安裝下列程式庫，包括任何相依性，並將其提供給 ggc_user 執行 ML 元件的系統使用者，例如。

- [Python](#) 3.8 或更新版本，包括 pip 適用於您 Python 版本的
- [TensorFlow Lite](#) 2.5.0 版
- [NumPy](#)
- [OpenCV-Python](#)
- [AWS IoT Device SDK 適用於 Python 的 v2](#)
- [AWS 常見執行時間 \(CRT\) Python](#)
- [Picamera](#) (適用於 Raspberry Pi 裝置)
- [awscam 模組](#) (適用於 AWS DeepLens 裝置)

- libGL (適用於 Linux 裝置)

預設：true

用途

將此元件與設定為 `UseInstaller` 組態參數搭配使用 `true`，以在您的裝置上安裝 TensorFlow Lite 及其相依性。元件會在您的裝置上設定虛擬環境，其中包含 TensorFlow Lite 所需的 OpenCV 和 NumPy 程式庫。

Note

此元件中的安裝程式指令碼也會安裝最新版本的其他系統程式庫，這些程式庫是在您的裝置上設定虛擬環境並使用已安裝的機器學習架構所必需。這可能會升級裝置上的現有系統程式庫。檢閱下表，以取得此元件針對每個支援的作業系統安裝的程式庫清單。如果您想要自訂此安裝程序，請將 `UseInstaller` 組態參數設定為 `false`，並開發您自己的安裝程式指令碼。

平台	裝置系統上安裝的程式庫	安裝在虛擬環境中的程式庫
Armv7l	build-essential , cmake, ca-certificates , git	setuptools , wheel
Amazon Linux 2	mesa-libGL	無
Ubuntu	wget	無

當您部署推論元件時，此執行時間元件會先驗證您的裝置是否已安裝 TensorFlow Lite 及其相依性。如果沒有，則執行期元件會為您安裝它們。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.5.15	針對 Greengrass 核 2.12.5 版更新版本。
2.5.14	針對 Greengrass nucleus 2.12.0 版更新版本。
2.5.13	針對 Greengrass nucleus 2.11.0 版更新版本。
2.5.12	針對 Greengrass nucleus 2.10.0 版更新版本。
2.5.11	針對 Greengrass nucleus 2.9.0 版更新版本。
2.5.10	針對 Greengrass nucleus 2.8.0 版更新版本。
2.5.9	針對 Greengrass nucleus 2.7.0 版更新版本。
2.5.8	針對 Greengrass nucleus 2.6.0 版更新版本。

版本	變更
2.5.7	錯誤修正和改善 - 更新UseInstaller 安裝指令碼以安裝 libGL，該 libGL 預設無法在某些 Linux 平台上使用。 - 更新UseInstaller 安裝指令碼，以一律在此元件的虛擬環境中使用 Python 3.9。此變更有助於確保與其他程式庫的相容性。
2.5.6	錯誤修正和改善 - 更新此元件以安裝 TensorFlow Lite 2.5.0 (tflite-runtime-2.5.0.post1) 的最新修補程式，讓您可以將此元件與 Python 3.9 搭配使用。如果此元件無法安裝該修補程式，則會tflite-runtime-2.5.0 改為安裝。 - 更新此元件以重新安裝 TensorFlow Lite 及其相依性的現有安裝。此變更有助於確保核心裝置執行相容版本的 TensorFlow Lite 及其相依性。
2.5.5	新功能 - 新增對執行 Windows 之核心裝置的支援。 - 新增可用於在 Windows 核心裝置上設定推論結果資料夾的新WindowsML RootPath 組態參數。
2.5.4	新功能 新增新的UseInstaller 組態參數，可讓您停用此元件中的安裝指令碼。
2.5.3	針對 Greengrass nucleus 2.4.0 版更新版本。
2.5.2	針對 Greengrass nucleus 2.3.0 版更新版本。
2.5.1	針對 Greengrass nucleus 2.2.0 版更新版本。
2.5.0	初始版本。

Modbus-RTU 通訊協定轉接器

Modbus-RTU 通訊協定轉接器元件 (aws.greengrass.Modbus) 會從本機 Modbus RTU 裝置輪詢資訊。

若要向具有此元件的本機 Modbus RTU 裝置請求資訊，請將訊息發佈至此元件訂閱的主題。在訊息中，指定要傳送至裝置的 Modbus RTU 請求。然後，此元件會發佈包含 Modbus RTU 請求結果的回應。

Note

此元件提供與 Modbus RTU 通訊協定轉接器連接器 in AWS IoT Greengrass V1 類似的功能。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的 [Modbus RTU 通訊協定轉接器連接器](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [輸出資料](#)
- [Modbus RTU 請求和回應](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

類型

此元件是 Lambda 元件 (aws.greengrass.lambda)。 [Greengrass 核](#) 會使用 Lambda [啟動器元件執行此元件的 Lambda](#) 函數。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- AWS IoT Greengrass 核心裝置與 Modbus 裝置之間的實體連線。核心裝置必須透過序列連接埠實際連接到 Modbus RTU 網路，例如 USB 連接埠。
- 若要從此元件接收輸出資料，您必須在部署此元件時合併[舊版訂閱路由器元件](#) (aws.greengrass.LegacySubscriptionRouter) 的下列組態更新。此組態會指定此元件發佈回應的主題。

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"
    }
  }
}

```

- 以您使用 AWS 區域 的 取代##。
- 以此元件執行的 Lambda 函數版本取代 版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的配方。在 [AWS IoT Greengrass 主控台](#) 中開啟此元件的詳細資訊頁面，並尋找 Lambda 函數鍵/值對。此鍵/值對包含 Lambda 函數的名稱和版本。

Important

每次部署此元件時，都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保針對您部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱[建立部署](#)。

- 支援 Modbus-RTU 通訊協定轉接器在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#) 中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.11

下表列出此元件 2.1.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.16.0	硬式

相依性	相容版本	相依性類型
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.15.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.14.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.4 and 2.1.5

下表列出此元件 2.1.4 和 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.8 and 2.1.0

下表列出此元件 2.0.8 和 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	硬式
Lambda 啟動器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式
字交換服務	>=1.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

Note

此元件的預設組態包含 Lambda 函數參數。建議您只編輯下列參數，以在裝置上設定此元件。

v2.1.x

lambdaParams

包含此元件 Lambda 函數參數的物件。此物件包含下列資訊：

EnvironmentVariables

包含 Lambda 函數參數的物件。此物件包含下列資訊：

ModbusLocalPort

核心裝置上實體 Modbus 序列連接埠的絕對路徑，例如 /dev/ttyS2。

若要在容器中執行此元件，您必須將此路徑定義為元件可存取的系統裝置（在 containerParams.devices）。根據預設，此元件會在容器中執行。

 Note

此元件必須具有裝置的讀取/寫入存取權。

ModbusBaudRate

（選用）字串值，指定與本機 Modbus TCP 裝置進行序列通訊的傳輸速率。

預設：9600

ModbusByteSize

（選用）字串值，指定與本機 Modbus TCP 裝置進行序列通訊的位元組大小。選擇 5、7、6 或 8 位元。

預設：8

ModbusParity

（選用）用於驗證與本機 Modbus TCP 裝置進行序列通訊之資料完整性的同位模式。

- E – 以同位驗證資料完整性。
- O – 使用奇數同位驗證資料完整性。
- N – 請勿驗證資料完整性。

預設：N

ModbusStopBits

(選用) 字串值，指定表示與本機 Modbus TCP 裝置序列通訊中位元組結束的位元數。

預設：1

containerMode

(選用) 此元件的容器化模式。您可以從以下選項中選擇：

- GreengrassContainer – 元件會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

如果您指定此選項，則必須指定系統裝置（在 `containerParams.devices`）中，讓容器存取 Modbus 裝置。

- NoContainer – 元件不會在隔離的執行時間環境中執行。

預設：GreengrassContainer

containerParams

(選用) 包含此元件容器參數的物件。如果您 GreengrassContainer 為指定，元件會使用這些參數 containerMode。

此物件包含下列資訊：

memorySize

(選用) 要配置給元件的記憶體量（以 KB 為單位）。

預設為 512 MB (525,312 KB)。

devices

(選用) 物件，指定元件可在容器中存取的系統裝置。

Important

若要在容器中執行此元件，您必須指定您在 `ModbusLocalPort` 環境變數中設定的系統裝置。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

path

核心裝置上系統裝置的路徑。這必須與您為 設定的值具有相同的值ModbusLocalPort。

permission

(選用) 從容器存取系統裝置的許可。此值必須是 `rw`，指定元件具有系統裝置的讀取/寫入存取權。

預設：`rw`

addGroupOwner

(選用) 是否新增執行元件的系統群組做為系統裝置的擁有者。

預設：`true`

pubsubTopics

(選用) 物件，其中包含元件訂閱接收訊息的主題。您可以指定每個主題，以及元件是從 訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

type

(選用) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- `PUB_SUB` - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，主題不能包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- `IOT_CORE` - 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設：`PUB_SUB`

topic

(選用) 元件訂閱接收訊息的主題。如果您 IotCore 為指定 type，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

Example 範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 範例：組態合併更新 (無容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

lambdaParams

包含此元件 Lambda 函數參數的物件。此物件包含下列資訊：

EnvironmentVariables

包含 Lambda 函數參數的物件。此物件包含下列資訊：

ModbusLocalPort

核心裝置上實體 Modbus 序列連接埠的絕對路徑，例如 `/dev/ttyS2`。

若要在容器中執行此元件，您必須將此路徑定義為元件可存取的系統裝置（在 `containerParams.devices`）。根據預設，此元件會在容器中執行。

Note

此元件必須具有裝置的讀取/寫入存取權。

containerMode

（選用）此元件的容器化模式。您可以從以下選項中選擇：

- `GreengrassContainer` – 元件會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

如果您指定此選項，則必須指定系統裝置（在 `containerParams.devices`），讓容器存取 Modbus 裝置。

- `NoContainer` – 元件不會在隔離的執行時間環境中執行。

預設：`GreengrassContainer`

containerParams

（選用）包含此元件容器參數的物件。如果您 `GreengrassContainer` 為指定，元件會使用這些參數 `containerMode`。

此物件包含下列資訊：

memorySize

（選用）要配置給元件的記憶體量（以 KB 為單位）。

預設為 512 MB (525, 312 KB)。

devices

（選用）物件，指定元件可在容器中存取的系統裝置。

 Important

若要在容器中執行此元件，您必須指定您在ModbusLocalPort環境變數中設定的系統裝置。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

path

核心裝置上系統裝置的路徑。這必須與您為 設定的值具有相同的值ModbusLocalPort。

permission

(選用) 從容器存取系統裝置的許可。此值必須是 `rw`，指定元件具有系統裝置的讀取/寫入存取權。

預設：`rw`

addGroupOwner

(選用) 是否新增執行元件的系統群組做為系統裝置的擁有者。

預設：`true`

pubsubTopics

(選用) 物件，其中包含元件訂閱接收訊息的主題。您可以指定每個主題，以及元件是從 訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

type

(選用) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，主題不能包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- IOT_CORE – 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設：PUB_SUB

topic

(選用) 元件訂閱接收訊息的主題。如果您IotCore為指定 type，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

Example範例：組態合併更新（容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example範例：組態合併更新（無容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  }
}
```

```
  },  
  "containerMode": "NoContainer"  
}
```

輸入資料

此元件接受下列主題的 Modbus RTU 請求參數，並將 Modbus RTU 請求傳送至裝置。根據預設，此元件會訂閱本機發佈/訂閱訊息。如需如何從自訂元件發佈訊息至此元件的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。

預設主題（本機發佈/訂閱）：modbus/adapter/request

訊息接受下列屬性。輸入訊息必須是 JSON 格式。

request

要傳送之 Modbus RTU 請求的參數。

請求訊息的形狀取決於其代表的 Modbus RTU 請求類型。所有請求都需要下列屬性。

類型：object 包含下列資訊：

operation

要執行的操作名稱。例如，指定 ReadCoilsRequest 讀取 Modbus RTU 裝置上的線圈。如需支援操作的詳細資訊，請參閱 [Modbus RTU 請求和回應](#)。

類型：string

device

請求的目標裝置。

此值必須是介於 0 和 247 之間的整數。

類型：integer

要包含在請求中的其他參數取決於操作。此元件會處理 [循環備援檢查 \(CRC\)](#)，以驗證資料請求。

Note

如果您請求包含 address 屬性，則必須將其值指定為整數。例如 "address": 1。

id

請求的任意 ID。使用此屬性將輸入請求映射至輸出回應。當您指定此屬性時，元件會將回應物件中的 id 屬性設定為此值。

類型：string

Example範例輸入：讀取線圈請求

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

輸出資料

此元件預設會將回應發佈為下列 MQTT 主題的輸出資料。您必須將此主題指定為[舊版訂閱路由器元件](#)組態subject中的。如需如何在自訂元件中訂閱此主題訊息的詳細資訊，請參閱[發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設主題 (AWS IoT Core MQTT)：modbus/adapt~~er~~/response

回應訊息的形狀取決於請求操作和回應狀態。如需範例，請參閱[範例請求和回應](#)。

每個回應含有以下屬性：

response

Modbus RTU 裝置的回應。

類型：object包含下列資訊：

status

請求的狀態。狀態可以是下列其中一個值：

- Success – 請求有效，元件將請求傳送到 Modbus RTU 網路，而 Modbus RTU 網路傳回回應。

- Exception – 請求有效，元件將請求傳送至 Modbus RTU 網路，而 Modbus RTU 網路傳回例外狀況。如需詳細資訊，請參閱[回應狀態：例外](#)。
- No Response – 請求無效，元件在將請求傳送到 Modbus RTU 網路之前攔截到錯誤。如需詳細資訊，請參閱[回應狀態：沒有回應](#)。

operation

元件請求的操作。

device

元件傳送請求的裝置。

payload

Modbus RTU 裝置的回應。如果 status 是 No Response，則此物件僅包含描述錯誤的 error 屬性（例如 [Input/Output] No Response received from the remote unit）。

id

請求的 ID，您可以用來識別哪些回應對應到哪個請求。

Note

寫入操作的回應只是請求的回響。雖然寫入回應不包含有意義的資訊，但最好檢查回應的狀態，以查看請求是否成功。

Example範例輸出：成功

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example範例輸出：失敗

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "MyRequest"
}
```

如需更多範例，請參閱[範例請求和回應](#)。

Modbus RTU 請求和回應

此連接器接受 Modbus RTU 請求參數做為[輸入資料](#)，並發佈回應做為[輸出資料](#)。

以下是支援的常見操作。

請求中的作業名稱	回應中的函數代碼
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16

請求中的作業名稱	回應中的函數代碼
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

範例請求和回應

以下是受支援操作的範例請求和回應。

讀取線圈

請求範例：

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

讀取離散輸入

請求範例：

```
{
  "request": {
    "operation": "ReadDiscreteInputsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadDiscreteInputsRequest",
    "payload": {
      "function_code": 2,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

讀取保留註冊

請求範例：

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
```

```
"response": {
  "status": "success",
  "device": 1,
  "operation": "ReadHoldingRegistersRequest",
  "payload": {
    "function_code": 3,
    "registers": [20,30]
  }
},
"id" : "TestRequest"
}
```

讀取輸入註冊

請求範例：

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

寫入單一線圈

請求範例：

```
{
  "request": {
    "operation": "WriteSingleCoilRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
```

```
"response": {
  "status": "success",
  "device": 1,
  "operation": "WriteSingleCoilRequest",
  "payload": {
    "function_code": 5,
    "address": 1,
    "value": true
  }
},
"id" : "TestRequest"
}
```

寫入單一登錄

請求範例：

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

寫入多個線圈

請求範例：

```
{
  "request": {
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleCoilsRequest",
    "payload": {
      "function_code": 15,
      "address": 1,
      "count": 4
    }
  },
  "id" : "TestRequest"
}
```

寫入多個登錄

請求範例：

```
{
  "request": {
    "operation": "WriteMultipleRegistersRequest",
    "device": 1,
    "address": 1,
    "values": [20,30,10]
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "address": 1,
      "count": 3
    }
  },
  "id" : "TestRequest"
}
```



```
}
```

遮罩寫入註冊

請求範例：

```
{
  "request": {
    "operation": "MaskWriteRegisterRequest",
    "device": 1,
    "address": 1,
    "and_mask": 175,
    "or_mask": 1
  },
  "id": "TestRequest"
}
```

回應範例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    }
  },
  "id": "TestRequest"
}
```

讀取寫入多個登錄

請求範例：

```
{
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
```

```
"read_count": 2,  
"write_address": 3,  
"write_registers": [20,30,40]  
},  
"id": "TestRequest"  
}
```

回應範例：

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "ReadWriteMultipleRegistersRequest",  
    "payload": {  
      "function_code": 23,  
      "registers": [10,20,10,20]  
    }  
  },  
  "id" : "TestRequest"  
}
```

 Note

回應包含元件讀取的註冊。

回應狀態：例外

當請求格式有效但請求未順利完成時會發生例外。在此情況下，回應包含下列資訊：

- status 設定為 Exception。
- function_code 等於請求的函數碼 + 128。
- exception_code 包含例外碼。如需詳細資訊，請參閱 Modbus 例外碼。

範例：

```
{  
  "response": {  
    "status": "fail",
```

```
{
  "error_message": "Internal Error",
  "error": "Exception",
  "device": 1,
  "operation": "ReadCoilsRequest",
  "payload": {
    "function_code": 129,
    "exception_code": 2
  }
},
"id": "TestRequest"
}
```

回應狀態：沒有回應

此連接器會在 Modbus 請求上執行驗證檢查。例如，它檢查是否格式無效和遺漏欄位。如果驗證失敗，連接器不會傳送請求。而是傳回包含以下資訊的回應：

- status 設定為 No Response。
- error 包含錯誤原因。
- error_message 包含錯誤訊息。

範例：

```
{
  "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"
}
```

如果請求的目標是不存在的裝置，或 Modbus RTU 網路沒有作用，您可能會收到 ModbusIOException (使用「沒有回應」格式)。

```
{
  "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"
}
```

本機日誌檔案

此元件使用以下日誌檔案。

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
```

授權

此元件包含下列第三方軟體/授權：

- [pymodbus](#)/BSD 授權
- [pyserial](#)/BSD 授權

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.11	針對 Greengrass nucleus 2.15.0 版更新版本。
2.1.10	針對 Greengrass nucleus 2.14.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.13.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.5	錯誤修正和改善 修正 ReadDiscreteInput 操作的問題。
2.1.4	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.0	新功能 新增 ModbusBaudRate 、ModbusParity 、ModbusByteSize 和 ModbusStopBits 選項，您可以指定這些選項來設定與 Modbus RTU 裝置的序列通訊。
2.0.8	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

MQTT 橋接器

MQTT 橋接元件 (`aws.greengrass.clientdevices.mqtt.Bridge`) 會在用戶端裝置、本機 Greengrass 發佈/訂閱和 之間轉送 MQTT 訊息 AWS IoT Core。您可以使用此元件對自訂元件中用戶端裝置的 MQTT 訊息採取行動，並使用 同步用戶端裝置 AWS 雲端。

Note

用戶端裝置是連線至 Greengrass 核心裝置的本機 IoT 裝置，用於傳送 MQTT 訊息和要處理的資料。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

您可以使用此元件在下列訊息代理程式之間轉送訊息：

- Local MQTT – 本機 MQTT 代理程式會處理用戶端裝置和核心裝置之間的訊息。
- 本機發佈/訂閱 – 本機 Greengrass 訊息代理程式會處理核心裝置上的元件之間的訊息。如需如何在 Greengrass 元件中與這些訊息互動的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- AWS IoT Core – AWS IoT Core MQTT 中介裝置會處理 IoT 裝置和 AWS 雲端 目的地之間的訊息。如需如何在 Greengrass 元件中與這些訊息互動的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)

- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。Greengrass 核會在與核相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 如果您將核心裝置的 MQTT 代理程式元件設定為使用預設連接埠 8883 以外的連接埠，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為在代理程式操作所在的連接埠上連線。
- MQTT 橋接器元件支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.3.2

下表列出此元件 2.3.2 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.6.0	硬式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.5.0	硬式

2.2.5 and 2.2.6

下表列出此元件 2.2.5 和 2.2.6 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.5.0	硬式

2.2.3 and 2.2.4

下表列出此元件 2.2.3 和 2.2.4 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.4.0	硬式

2.2.0 – 2.2.2

下表列出此元件 2.2.0 至 2.2.2 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.3.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.0.0 <2.2.0	硬式

2.0.0 to 2.1.0

下表列出此元件 2.0.0 到 2.1.0 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.0.0 <2.1.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.3.0 – 2.3.2

mqttTopicMapping

您要橋接的主題映射。此元件會訂閱來源主題的訊息，並將收到的訊息發佈至目的地主題。每個主題映射都會定義主題、來源類型和目的地類型。

此物件包含下列資訊：

topicMappingNameKey

此主題映射的名稱。將 *topicMappingNameKey* 取代為可協助您識別此主題映射的名稱。

此物件包含下列資訊：

topic

要橋接來源和目標代理程式的主題或主題篩選條件。

您可以使用 + 和 # MQTT 主題萬用字元，在所有符合主題篩選條件的主題上轉送訊息。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

 Note

若要搭配 Pubsub 來源代理程式使用 MQTT 主題萬用字元，您必須使用 [Greengrass 核元件](#) 的 v2.6.0 或更新版本。

targetTopicPrefix

當此元件轉送訊息時，要新增至目標主題的字首。

source

來源訊息代理程式。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

 Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT

代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

target

目標訊息中介裝置。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

 Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

mqtt5RouteOptions

(選用) 提供設定主題映射的選項，以將訊息從來源主題橋接至目的地主題。

此物件包含下列資訊：

mqtt5RouteOptionsNameKey

主題映射的路由選項名稱。將 *mqtt5RouteOptionsNameKey* 取代為 mqttTopicMapping 欄位中定義的相符 *topicMappingNameKey*。

此物件包含下列資訊：

noLocal

(選用) 啟用時，橋接器不會轉送橋接器本身發佈之主題的訊息。使用此選項來防止迴圈，如下所示：

```
{
  "mqtt5RouteOptions": {
    "toIoTCore": {
      "noLocal": true
    }
  },
  "mqttTopicMapping": {
    "toIoTCore": {
      "topic": "device",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "toLocal": {
      "topic": "device",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

`noLocal` 僅支援 `source` 為 的路由 `LocalMqtt`。

預設：false

`retainAsPublished`

（選用）啟用時，橋接器轉送的訊息具有與該路由發佈至代理程式的訊息相同的 `retain` 旗標。

`retainAsPublished` 僅支援 `source` 為 的路由 `LocalMqtt`。

預設：false

`mqtt`

（選用）用於與本機代理程式通訊的 MQTT 通訊協定設定。

`version`

（選用）橋接器用來與本機代理程式通訊的 MQTT 通訊協定版本。必須與核心組態中選取的 MQTT 版本相同。

請選擇下列項目：

- `mqtt3`

- mqtt5

當mqttTopicMapping物件的 source或 target 欄位設定為 時，您必須部署 MQTT 代理程式LocalMqtt。如果您選擇 mqtt5選項，則必須使用 [MQTT 5 代理程式 \(EMQX\)](#)。

預設：mqtt3

ackTimeoutSeconds

(選用) 在操作失敗之前等待 PUBACK、SUBACK 或 UNSUBACK 封包的時間間隔。

預設：60

connAckTimeoutMs

(選用) 關閉連線之前等待 CONNACK 封包的時間間隔。

預設：20000 (20 秒)

pingTimeoutMs

(選用) 橋接器等待從本機代理程式接收 PINGACK 訊息的時間，以毫秒為單位。如果等待超過逾時，橋接會關閉，然後重新開啟 MQTT 連線。此值必須小於 keepAliveTimeoutSeconds。

預設：30000 (30 秒)

keepAliveTimeoutSeconds

(選用) 橋接器為保持 MQTT 連線持續傳送的每個 PING 訊息之間的秒數。此值必須大於 pingTimeoutMs。

預設：60

maxReconnectDelayMs

(選用) MQTT 重新連線的秒數上限。

預設：30000 (30 秒)

minReconnectDelayMs

(選用) MQTT 重新連線的最短秒數。

receiveMaximum

(選用) 橋接器可以傳送的未確認 QoS1 封包數目上限。

預設：100

maximumPacketSize

用戶端接受 MQTT 封包的最大位元組數。

預設：null（無限制）

sessionExpiryInterval

（選用）您可以請求工作階段在橋接器與本機代理程式之間持續的時間，以秒為單位。

預設：4294967295（工作階段永不過期）

brokerUri

（選用）本機 MQTT 代理程式的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。使用下列格式，並將###取代為 MQTT 中介裝置操作的連接埠：`ssl://localhost:port`。

預設：`ssl://localhost:8883`

startupTimeoutSeconds

（選用）元件啟動的秒數上限。BROKEN 如果元件超過此逾時，元件的狀態會變更為。

預設：120

Example範例：組態合併更新

下列範例組態更新會指定下列項目：

- 在符合主題篩選條件`clients/+ /hello/world`的主題 AWS IoT Core 上，將訊息從用戶端裝置轉送至。
- 將訊息從用戶端裝置轉送至本機發佈/訂閱符合`clients/+ /detections`主題篩選條件的主題，並將`events/input/`字首新增至目標主題。產生的目標主題符合`events/input/clients/+ /detections`主題篩選條件。
- 在符合主題篩選條件`clients/+ /status`的主題 AWS IoT Core 上，將訊息從用戶端裝置轉送至，並將`$aws/rules/StatusUpdateRule/`字首新增至目標主題。此範例會將這些訊息直接轉送至名為的 [AWS IoT 規則](#) `StatusUpdateRule`，以使用 [基本擷取](#) 降低成本。

```
{
```

```

"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範例：設定 MQTT 5

下列範例組態會更新下列項目：

- 讓橋接器搭配本機代理程式使用 MQTT 5 通訊協定。
- 將 MQTT 保留設定為ClientDeviceHelloWorld主題映射的已發佈設定。

```

{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {

```

```
    "version": "mqtt5"
  }
}
```

2.2.6

mqttTopicMapping

您要橋接的主題映射。此元件會訂閱來源主題的訊息，並將收到的訊息發佈至目的地主題。每個主題映射都會定義主題、來源類型和目的地類型。

此物件包含下列資訊：

topicMappingNameKey

此主題映射的名稱。將 *topicMappingNameKey* 取代為可協助您識別此主題映射的名稱。

此物件包含下列資訊：

topic

要橋接來源和目標代理程式的主題或主題篩選條件。

您可以使用 + 和 # MQTT 主題萬用字元，在所有符合主題篩選條件的主題上轉送訊息。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

Note

若要搭配 Pubsub 來源代理程式使用 MQTT 主題萬用字元，您必須使用 [Greengrass 核元件](#) 的 v2.6.0 或更新版本。

targetTopicPrefix

當此元件轉送訊息時，要新增至目標主題的字首。

source

來源訊息代理程式。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

target

目標訊息中介裝置。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

brokerUri

(選用) 本機 MQTT 代理程式的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。使用下列格式，並將###取代為 MQTT 中介裝置操作所在的連接埠：`ssl://localhost:port`。

預設：`ssl://localhost:8883`

startupTimeoutSeconds

(選用) 元件啟動的秒數上限。BROKEN 如果元件超過此逾時，元件的狀態會變更為。

預設：120

Example範例：組態合併更新

下列範例組態更新會指定下列項目：

- 在符合主題篩選條件clients/+/hello/world的主題 AWS IoT Core 上，將訊息從用戶端裝置轉送至。
- 在符合主題篩選條件clients/+/detections的主題上，將訊息從用戶端裝置轉送至本機發佈/訂閱，並將events/input/字首新增至目標主題。產生的目標主題符合events/input/clients/+/detections主題篩選條件。
- 在符合主題篩選條件clients/+/status的主題 AWS IoT Core 上，將訊息從用戶端裝置轉送到，並將\$aws/rules/StatusUpdateRule/字首新增至目標主題。此範例會將這些訊息直接轉送至名為的[AWS IoT 規則](#)StatusUpdateRule，以使用[基本擷取](#)降低成本。

```
{
  "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

您要橋接的主題映射。此元件會訂閱來源主題的訊息，並將收到的訊息發佈至目的地主題。每個主題映射都會定義主題、來源類型和目的地類型。

此物件包含下列資訊：

topicMappingNameKey

此主題映射的名稱。將 *topicMappingNameKey* 取代為可協助您識別此主題映射的名稱。

此物件包含下列資訊：

topic

要橋接來源和目標代理程式的主題或主題篩選條件。

您可以使用 + 和 # MQTT 主題萬用字元，在符合主題篩選條件的所有主題上轉送訊息。
如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

 Note

若要搭配 Pubsub 來源代理程式使用 MQTT 主題萬用字元，您必須使用 [Greengrass 核元件](#) 的 v2.6.0 或更新版本。

targetTopicPrefix

當此元件轉送訊息時，要新增至目標主題的字首。

source

來源訊息代理程式。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

 Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

target

目標訊息中介裝置。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

brokerUri

(選用) 本機 MQTT 中介裝置的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。使用下列格式，並將###取代為 MQTT 中介裝置操作所在的連接埠：`ssl://localhost:port`。

預設：`ssl://localhost:8883`

Example範例：組態合併更新

下列範例組態更新會指定下列項目：

- 在符合主題篩選條件`clients/+ /hello/world`的主題 AWS IoT Core 上，將訊息從用戶端裝置轉送至。
- 在符合主題篩選條件`clients/+ /detections`的主題上，將訊息從用戶端裝置轉送至本機發佈/訂閱，並將`events/input/`字首新增至目標主題。產生的目標主題符合`events/input/clients/+ /detections`主題篩選條件。
- 在符合主題篩選條件`clients/+ /status`的主題 AWS IoT Core 上，將訊息從用戶端裝置轉送到，並將`$aws/rules/StatusUpdateRule/`字首新增至目標主題。此範例會將這些訊息直接轉送至名為的 [AWS IoT 規則](#) `StatusUpdateRule`，以使用 [基本擷取](#) 降低成本。

```
{
  "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

您要橋接的主題映射。此元件會訂閱來源主題的訊息，並將收到的訊息發佈至目的地主題。每個主題映射都會定義主題、來源類型和目的地類型。

此物件包含下列資訊：

topicMappingNameKey

此主題映射的名稱。將 *topicMappingNameKey* 取代為可協助您識別此主題映射的名稱。

此物件包含下列資訊：

topic

要橋接來源和目標代理程式的主題或主題篩選條件。

如果您指定 LocalMqtt 或 IotCore 來源代理程式，您可以使用 + 和 # MQTT 主題萬用字元，在所有符合主題篩選條件的主題上轉送訊息。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

source

來源訊息代理程式。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

target

目標訊息中介裝置。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

brokerUri

(選用) 本機 MQTT 中介裝置的 URI。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。使用下列格式，並將 **###** 取代為 MQTT 中介裝置操作所在的連接埠：`ssl://localhost:port`。

預設：`ssl://localhost:8883`

Example範例：組態合併更新

下列範例組態更新會指定在 `clients/MyClientDevice1/hello/world` 和 `clients/MyClientDevice2/hello/world` 主題 AWS IoT Core 上，將訊息從用戶端裝置轉送至。

```
{
  "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

您要橋接的主題映射。此元件會訂閱來源主題的訊息，並將收到的訊息發佈至目的地主題。每個主題映射都會定義主題、來源類型和目的地類型。

此物件包含下列資訊：

topicMappingNameKey

此主題映射的名稱。將 *topicMappingNameKey* 取代為可協助您識別此主題映射的名稱。

此物件包含下列資訊：

topic

要橋接來源和目標代理程式的主題或主題篩選條件。

如果您指定 LocalMqtt 或 IotCore 來源代理程式，您可以使用 + 和 # MQTT 主題萬用字元，在所有符合主題篩選條件的主題上轉送訊息。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

source

來源訊息代理程式。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

target

目標訊息中介裝置。您可以從以下選項中選擇：

- LocalMqtt – 用戶端裝置通訊的本機 MQTT 代理程式。
- Pubsub – 本機 Greengrass 發佈/訂閱訊息代理程式。
- IotCore – AWS IoT Core MQTT 訊息代理程式。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送 MQTT 訊息時，您可能會發現額外的延遲 AWS

IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

source 和 target 必須不同。

Example範例：組態合併更新

下列範例組態更新會指定在 clients/MyClientDevice1/hello/world和 clients/MyClientDevice2/hello/world主題 AWS IoT Core 上，將訊息從用戶端裝置轉送至。

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#)元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.3.2	針對 用戶端裝置驗證 2.5.0 版更新版本。
2.3.1	錯誤修正和改善 修正本機 MQTT 用戶端進入中斷連線迴圈的問題。
2.3.0	新功能 新增 MQTT5 支援在 AWS IoT Core 和本機 MQTT 來源之間橋接。
2.2.6	新功能 新增 <code>startupTimeoutSeconds</code> 組態選項。
2.2.5	針對 用戶端裝置驗證 2.4.0 版更新版本。
2.2.4	針對 Greengrass 用戶端裝置驗證 2.3.0 版更新版本。
2.2.3	此版本包含錯誤修正和改善。

版本	變更
2.2.2	錯誤修正和改善 記錄調整。
2.2.1	錯誤修正和改善 修正可能導致 MQTT 橋接器無法訂閱 MQTT 主題的問題。
2.2.0	新功能 - 當您將本機發佈/訂閱指定為來源訊息代理程式時，新增對 MQTT 主題萬用字元 (# 和 +) 的支援。 此功能需要 2.6.0 版或更新版本的 Greengrass 核心元件。 - 新增 選項，您可以指定此targetTopicPrefix 選項來設定 MQTT 橋接器，以在轉送訊息時將字首新增至目標主題。
2.1.1	錯誤修正和改善 - 修正此元件如何處理組態重設更新的問題。 - 降低憑證輪換時 MQTT 用戶端中斷連線的頻率。
2.1.0	新功能 新增 brokerUri 參數，可讓您使用非預設的 MQTT 代理程式連接埠。
2.0.1	此版本包含錯誤修正和改善。
2.0.0	初始版本。

MQTT 3.1.1 代理程式 (Moquette)

Moquette MQTT 代理程式元件 (aws.greengrass.clientdevices.mqtt.Moquette) 會處理用戶端裝置和 Greengrass 核心裝置之間的 MQTT 訊息。此元件提供修改版本的 [Moquette MQTT 代理程式](#)。部署此 MQTT 代理程式以執行輕量型 MQTT 代理程式。如需如何選擇 MQTT 代理程式的詳細資訊，請參閱 [選擇 MQTT 代理程式](#)。

此代理程式實作 MQTT 3.1.1 通訊協定。它包含對 QoS 0、QoS 1、QoS 2 保留訊息、最後遺囑訊息和持久性工作階段的支援。

 Note

用戶端裝置是連線至 Greengrass 核心裝置的本機 IoT 裝置，用於傳送 MQTT 訊息和要處理的資料。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 核心裝置必須能夠在 MQTT 中介裝置操作的連接埠上接受連線。此元件預設會在連接埠 8883 上執行 MQTT 代理程式。您可以在設定此元件時指定不同的連接埠。

如果您指定不同的連接埠，並使用 [MQTT 橋接器元件](#) 將 MQTT 訊息轉送至其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 中介裝置操作所在的連接埠。

如果您指定不同的連接埠，並使用 [IP 偵測器元件](#) 來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 中介裝置操作所在的連接埠。

- 支援在 VPC 中執行 Moquette MQTT 代理程式元件。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.3.7

下表列出此元件 2.3.7 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.6.0	硬式

2.3.2 – 2.3.6

下表列出此元件 2.3.2 至 2.3.6 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.5.0	硬式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.4.0	硬式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.3.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.0.0 <2.2.0	硬式

2.0.0 - 2.0.2

下表列出此元件 2.0.0 到 2.0.2 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.0.0 <2.1.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

moquette

(選用) 要使用的 [Moquette MQTT 代理程式](#) 組態。您可以在此元件中設定 Moquette 組態選項的子集。如需詳細資訊，請參閱 [Moquette 組態檔案中](#) 的內嵌註解。

此物件包含下列資訊：

ssl_port

(選用) MQTT 中介裝置操作所在的連接埠。

Note

如果您指定不同的連接埠，並使用 [MQTT 橋接器元件](#) 將 MQTT 訊息轉送至其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 中介裝置操作所在的連接埠。

如果您指定不同的連接埠，並使用 [IP 偵測器元件](#) 來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 中介裝置操作所在的連接埠。

預設：8883

host

(選用) MQTT 中介裝置繫結的介面。例如，您可以變更此參數，以便 MQTT 代理程式僅繫結至特定本機網路。

預設：0.0.0.0 (繫結至所有網路介面)

startupTimeoutSeconds

(選用) 元件啟動的時間上限，以秒為單位。BROKEN 如果元件超過此逾時，元件的狀態會變更為。

預設：120

Example範例：組態合併更新

下列範例組態指定 在連接埠 443 上操作 MQTT 代理程式。

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```


Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.3.7	針對 用戶端裝置驗證 2.5.0 版更新版本。
2.3.6	錯誤修正和改善 一般錯誤修正與改進。
2.3.5	錯誤修正和改善 將 Moquette 更新至 0.17 版。
2.3.4	錯誤修正和改善 修正用戶端在傳送或接收訊息時可能因為重複IDs 而遇到無效工作階段錯誤的問題。此問題導致用戶端的工作階段關閉。
2.3.3	新功能 新增startupTimeoutSeconds 組態選項。
2.3.2	針對 用戶端裝置驗證 2.4.0 版更新版本。
2.3.1	錯誤修正和改善 修正由於工作階段無效，用戶端可能會在嘗試重新連線後中斷連線的競爭條件。
2.3.0	新增憑證鏈的支援。
2.2.0	針對 用戶端裝置驗證 2.2.0 版更新版本。

版本	變更
2.1.0	錯誤修正和改善 - 更新此元件以使用 Moquette 0.16 版，這可改善效能並包含數個其他改進項目。 - 修正本機 MQTT 伺服器憑證在特定情況下旋轉頻率高於預期的問題。 若要套用此修正，您還必須使用 v2.1.0 或更新版本的用戶端裝置身分驗證元件。
2.0.2	錯誤修正和改善 - 將 MQTT 訊息大小上限從 8,092 位元組增加到 128 KB。有效的 MQTT 訊息承載限制略低，因為訊息大小限制包含訊息標頭。 - 在 <code>ssl_port</code> 參數中新增整數值的支援。
2.0.1	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.0	初始版本。

MQTT 5 代理程式 (EMQX)

EMQX MQTT 代理程式元件 (`aws.greengrass.clientdevices.mqtt.EMQX`) 會處理用戶端裝置和 Greengrass 核心裝置之間的 MQTT 訊息。此元件提供修改版本的 [EMQX MQTT 5.0 代理程式](#)。部署此 MQTT 代理程式以使用 MQTT 5 功能在用戶端裝置和核心裝置之間進行通訊。如需如何選擇 MQTT 代理程式的詳細資訊，請參閱 [選擇 MQTT 代理程式](#)。

此代理程式實作 MQTT 5.0 通訊協定。它包含對工作階段和訊息過期間隔、使用者屬性、共用訂閱、主題別名等的支援。MQTT 5 與 MQTT 3.1.1 回溯相容，因此如果您執行 [Moquette MQTT 3.1.1 代理程式](#)，您可以將其取代為 EMQX MQTT 5 代理程式，而且用戶端裝置可以繼續照常連線和操作。

Note

用戶端裝置是連線至 Greengrass 核心裝置的本機 IoT 裝置，用於傳送 MQTT 訊息和要處理的資料。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.0.x
- 1.2.x
- 1.1.x
- 1.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- 核心裝置必須能夠在 MQTT 中介裝置操作的連接埠上接受連線。此元件預設會在連接埠 8883 上執行 MQTT 代理程式。您可以在設定此元件時指定不同的連接埠。

如果您指定不同的連接埠，並使用 [MQTT 橋接器元件](#) 將 MQTT 訊息轉送至其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 代理程式操作所在的連接埠。

如果您指定不同的連接埠，並使用 [IP 偵測器元件](#) 來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 中介裝置操作所在的連接埠。

- 在 Linux 核心裝置上，Docker 已安裝並設定在核心裝置上：
 - [Docker 引擎](#) 1.9.1 或更新版本安裝在 Greengrass 核心裝置上。20.10 版是經過驗證可與 AWS IoT Greengrass Core 軟體搭配使用的最新版本。您必須先直接在核心裝置上安裝 Docker，才能部署執行 Docker 容器的元件。
 - 在部署此元件之前，Docker 協助程式會在核心裝置上啟動和執行。
 - 執行此元件的系統使用者必須具有根或管理員許可。或者，您可以在 docker 群組中以系統使用者身分執行此元件，並將此元件 requiresPrivileges 的選項設定為 false，以在沒有權限的情況下執行 EMQX MQTT 代理程式。
- EMQX MQTT 代理程式元件支援在 VPC 中執行。
- armv7 平台不支援 EMQX MQTT 代理程式元件。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件 [發行版本的](#) 相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以在此 [AWS IoT Greengrass 主控台](#) 中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.0.2 – 2.0.3

下表列出此元件 2.0.2 和 2.0.3 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	>=2.2.0 <2.6.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	$\geq 2.2.0 < 2.6.0$	硬式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	$\geq 2.2.0 < 2.5.0$	硬式

1.2.2 – 1.2.3

下表列出此元件 1.2.2 至 1.2.3 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	$\geq 2.2.0 < 2.5.0$	硬式

1.2.0 and 1.2.1

下表列出此元件 1.2.0 和 1.2.1 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	$\geq 2.2.0 < 2.4.0$	硬式

1.0.0 and 1.1.0

下表列出此元件 1.0.0 和 1.1.0 版的相依性。

相依性	相容版本	相依性類型
用戶端裝置身分驗證	$\geq 2.2.0 < 2.3.0$	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

2.0.0 - 2.0.3

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

Important

如果您使用 MQTT 5 代理程式 (EMQX) 元件的第 2 版，則必須更新組態檔案。第 1 版組態檔案不適用於第 2 版。

emqxConfig

(選用) 要使用的 [EMQX MQTT 代理程式](#) 組態。您可以在此元件中設定 EMQX 組態選項。

當您使用 EMQX 代理程式時，Greengrass 會使用預設組態。除非您使用此欄位修改組態，否則使用此組態。

修改下列組態設定會導致 EMQX 代理程式元件重新啟動。其他組態變更會套用而不重新啟動元件。

- emqxConfig/cluster
- emqxConfig/node
- emqxConfig/rpc

Note

`aws.greengrass.clientdevices.mqtt.EMQX` 可讓您設定安全敏感選項。這些包括 TLS 設定、身分驗證和授權提供者。我們建議使用交互 TLS 身分驗證和 Greengrass 用戶端裝置身分驗證提供者的預設組態。

Example範例：預設組態

下列範例顯示為 MQTT 5 (EMQX) 代理程式設定的預設值。您可以使用組態設定覆寫這些 emqxConfig 設定。

```
{
```

```
"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

(選用) 設定代理程式的授權提供者。可以是下列其中一個值：

- enabled – (預設) 使用 Greengrass 身分驗證和授權提供者。
- bypass_on_failure – 如果 Greengrass 拒絕身分驗證或授權，請使用 Greengrass 身分驗證提供者，然後使用 EMQX 提供者鏈結中任何剩餘的身分驗證提供者。
- bypass – Greengrass 提供者已停用。身分驗證和授權是由 EMQX 供應商鏈處理。

requiresPrivilege

(選用) 在 Linux 核心裝置上，您可以指定在沒有根或管理員權限的情況下執行 EMQX MQTT 代理程式。如果您將此選項設定為 false，則執行此元件的系統使用者必須是 docker 群組的成員。

預設：true

startupTimeoutSeconds

(選用) EMQX MQTT 代理程式啟動的秒數上限。BROKEN 如果元件超過此逾時，元件的狀態會變更為。

預設：90

ipcTimeoutSeconds

(選用) 元件等待 Greengrass 核回應程序間通訊 (IPC) 請求的秒數上限。如果此元件在檢查用戶端裝置是否獲得授權時回報逾時錯誤，請增加此數字。

預設：5

crtLogLevel

(選用) AWS 一般執行期 (CRT) 程式庫的日誌層級。

預設為 EMQX MQTT 代理程式日誌層級 (log.level 中的 emqx)。

restartIdentifier

(選用) 設定此選項以重新啟動 EMQX MQTT 代理程式。當此組態值變更時，此元件會重新啟動 MQTT 代理程式。您可以使用此選項來強制用戶端裝置中斷連線。

dockerOptions

(選用) 僅在 Linux 作業系統上設定此選項，將參數新增至 Docker 命令列。例如，若要映射其他連接埠，請使用 -p Docker 參數：


```
"-p 1883:1883"
```

Example範例：將 v1.x 組態檔案更新為 v2.x

下列範例顯示將 v1.x 組態檔案更新為 2.x 版所需的變更。

1.x 版組態檔案：

```
{
  "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"
  }
}
```

v2 的同等組態檔案：

```
{
  "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"  
  }  
}
```

組態 `listener.ssl.external.rate_limit` 項目沒有同等項目。 `use_greengrass_managed_certificates` 組態選項已移除。

Example 範例：設定代理程式的新連接埠

下列範例會將 MQTT 中介裝置從預設 8883 操作的連接埠變更為連接埠 1234。如果您使用的是 Linux，請包含 `dockerOptions` 欄位。

```
{  
  "emqxConfig": {  
    "listeners": {  
      "ssl": {  
        "default": {  
          "bind": 1234  
        }  
      }  
    }  
  },  
  "dockerOptions": "-p 1234:1234"  
}
```

Example 範例：調整 MQTT 代理程式的日誌層級

下列範例會將 MQTT 代理程式的日誌層級變更為 `debug`。您可以從下列日誌層級中選擇：

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`

- emergency

預設日誌層級為 warning。

```
{
  "emqxConfig": {
    "log": {
      "console": {
        "level": "debug"
      }
    }
  }
}
```

Example範例：啟用 EMQX 儀表板

下列範例會啟用 EMQX 儀表板，讓您可以監控和管理代理程式。如果您使用的是 Linux，請包含 dockerOptions 欄位。

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

emqx

(選用) 要使用的 [EMQX MQTT 代理程式](#) 組態。您可以在此元件中設定 EMQX 組態選項的子集。

此物件包含下列資訊：

`listener.ssl.external`

(選用) MQTT 中介裝置操作所在的連接埠。

 Note

如果您指定不同的連接埠，並使用 [MQTT 橋接器元件](#) 將 MQTT 訊息轉送給其他代理程式，則必須使用 MQTT 橋接器 v2.1.0 或更新版本。將其設定為使用 MQTT 代理程式操作所在的連接埠。

如果您指定不同的連接埠，並使用 [IP 偵測器元件](#) 來管理 MQTT 代理程式端點，則必須使用 IP 偵測器 v2.1.0 或更新版本。將其設定為報告 MQTT 中介裝置操作所在的連接埠。

預設：8883

`listener.ssl.external.max_connections`

(選用) MQTT 中介裝置支援的並行連線數目上限。

預設：1024000

`listener.ssl.external.max_conn_rate`

(選用) MQTT 代理程式每秒可接收的新連線數上限。

預設：500

`listener.ssl.external.rate_limit`

(選用) 所有 MQTT 代理程式連線的頻寬限制。指定該頻寬的頻寬和持續時間，並以逗號(,) 分隔，格式如下：`bandwidth,duration`。例如，您可以指定 `50KB,5s` 將 MQTT 代理程式限制為每 5 秒 50 KB (KB) 的資料。

`listener.ssl.external.handshake_timeout`

(選用) MQTT 代理程式等待完成驗證新連線的時間長度。

預設：15s

`mqtt.max_packet_size`

(選用) MQTT 訊息的大小上限。

預設：268435455(256 MB 減去 1)

log.level

(選用) MQTT 代理程式的日誌層級。您可以從以下選項中選擇：

- debug
- info
- notice
- warning
- error
- critical
- alert
- emergency

預設日誌層級為 warning。

requiresPrivilege

(選用) 在 Linux 核心裝置上，您可以指定在沒有根或管理員權限的情況下執行 EMQX MQTT 代理程式。如果您將此選項設定為 false，則執行此元件的系統使用者必須是 docker 群組的成員。

預設：true

startupTimeoutSeconds

(選用) EMQX MQTT 代理程式啟動的秒數上限。BROKEN 如果元件超過此逾時，元件的狀態會變更為。

預設：90

ipcTimeoutSeconds

(選用) 元件等待 Greengrass 核回應程序間通訊 (IPC) 請求的秒數上限。如果此元件在檢查用戶端裝置是否獲得授權時回報逾時錯誤，請增加此數字。

預設：5

crtLogLevel

(選用) AWS 一般執行期 (CRT) 程式庫的日誌層級。

預設為 EMQX MQTT 代理程式日誌層級 (log.level 中的 emqx)。

restartIdentifier

(選用) 設定此選項以重新啟動 EMQX MQTT 代理程式。當此組態值變更時，此元件會重新啟動 MQTT 代理程式。您可以使用此選項來強制用戶端裝置中斷連線。

dockerOptions

(選用) 僅在 Linux 作業系統上設定此選項，將參數新增至 Docker 命令列。例如，若要映射其他連接埠，請使用 `-p Docker 參數`：

```
"-p 1883:1883"
```

mergeConfigurationFiles

(選用) 設定此選項以新增或覆寫指定 EMQX 組態檔案中的預設值。如需有關組態檔案及其格式的資訊，請參閱 EMQX 4.0 文件中的[組態](#)。您指定的值會附加到組態檔案。

下列範例會更新 `etc/emqx.conf` 檔案。

```
"mergeConfigurationFiles": {  
  "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"  
},
```

除了 EMQX 支援的組態檔案之外，Greengrass 還支援名為 `etc/plugins/aws_greengrass_emqx_auth.conf` 的 EMQX 設定 Greengrass 驗證外掛程式的檔案。支援兩種選項：`auth_mode` 和 `use_greengrass_managed_certificates`。若要使用另一個身分驗證提供者，請將 `auth_mode` 選項設定為下列其中一項：

- `enabled` – (預設) 使用 Greengrass 身分驗證和授權提供者。
- `bypass_on_failure` – 如果 Greengrass 拒絕身分驗證或授權，請使用 Greengrass 身分驗證提供者，然後使用 EMQX 提供者鏈結中任何剩餘的身分驗證提供者。
- `bypass` – Greengrass 提供者已停用。然後，EMQX 供應商鏈會處理身分驗證和授權。

如果 `use_greengrass_managed_certificates` 是 `true`，此選項表示 Greengrass 管理代理程式 TLS 憑證。如果為 `false`，表示您透過另一個來源提供憑證。

下列範例會更新 `etc/plugins/aws_greengrass_emqx_auth.conf` 組態檔案中的預設值。

```
"mergeConfigurationFiles": {  
  "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\nuse_greengrass_managed_certificates=true\n"  
},
```

 Note

`aws.greengrass.clientdevices.mqtt.EMQX` 可讓您設定安全敏感選項。這些包括 TLS 設定、身分驗證和授權提供者。建議組態是使用交互 TLS 身分驗證和 Greengrass 用戶端裝置身分驗證提供者的預設組態。

replaceConfigurationFiles

(選用) 設定此選項以取代指定的 EMQX 組態檔案。您指定的值會取代整個現有的組態檔案。您無法在本節中指定 `etc/emqx.conf` 檔案。您必須使用 `mergeConfigurationFile` 來修改 `etc/emqx.conf`。

Example範例：組態合併更新

下列範例組態指定 在連接埠 443 上操作 MQTT 代理程式。

```
{
  "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"
}
```

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

授權

在 Windows 作業系統上，本軟體包含根據 [Microsoft 軟體授權條款 - Microsoft Visual Studio Community 2022](#) 分佈的程式碼。下載本軟體即表示您同意該程式碼的授權條款。

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

v2.x

版本	變更
2.0.3	錯誤修正和改善 修正如果路徑包含空格，EMQX 不會在 Windows 上啟動的問題。
2.0.2	錯誤修正和改善 修正 EMQX 在用戶端裝置驗證元件就緒之前啟動的問題。

版本	變更
2.0.1	針對 用戶端裝置驗證 2.5.0 版更新版本。
2.0.0	此版本的 MQTT 5 代理程式 (EMQX) 預期與 1.x 版不同的組態參數。如果您使用 1.x 版的非預設組態，則必須更新 2.x 的元件組態。如需詳細資訊，請參閱Configuration。 新功能 - 將 MQTT 代理程式升級至 EMQX 5.1.1。 - 啟用代理程式組態變更，而不重新啟動元件。 更新 - 新增取代 <code>emqx</code>、<code>mergeConfigurationFiles</code> 和 <code>emqxConfig</code> 組態欄位的新 <code>replaceConfigurationFiles</code> 組態欄位。

v1.x

版本	變更
1.2.3	錯誤修正和改善 修正用戶端在先前透過中斷連線並重新驗證用戶端進行身分驗證之後，無法與 EMQX 互動的問題。
1.2.2	針對 用戶端裝置驗證 2.4.0 版更新版本。
1.2.1	錯誤修正和改善 - 修正如果 Visual C++ 可轉散發套件尚未存在，元件無法在 Windows 上啟動的問題。 - 將 EMQX 更新至 4.4.14 版。
1.2.0	新增憑證鏈的支援。
1.1.0	新功能 新增對 EMQX 組態的支援，包括代理程式選項和外掛程式。

版本	變更
	錯誤修正和改善 將 EMQX 更新至 4.4.9 版。
1.0.1	修正 TLS 交握期間導致某些 MQTT 用戶端無法連線的問題。
1.0.0	初始版本。

Nucleus 遙測發射器

核遙測發射器元件 (`aws.greengrass.telemetry.NucleusEmitter`) 會收集系統運作狀態遙測資料，並將其持續發佈至本機主題和 AWS IoT Core MQTT 主題。此元件可讓您在 Greengrass 核心裝置上收集即時系統遙測。如需將系統遙測資料發佈至 Amazon EventBridge 的 Greengrass 遙測代理程式相關資訊，請參閱 [從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)。

根據預設，核遙測發射器元件會每 60 秒將遙測資料發佈至下列本機發佈/訂閱主題。

```
$local/greengrass/telemetry
```

根據預設，核遙測發射器元件不會發佈至 AWS IoT Core MQTT 主題。您可以設定此元件在部署時發佈至 AWS IoT Core MQTT 主題。使用 MQTT 主題將資料發佈到 AWS 雲端 需受 [AWS IoT Core 定價](#) 的限制。

AWS IoT Greengrass 提供多種 [社群元件](#)，可協助您使用 InfluxDB 和 Grafana 在本機核心裝置上分析和視覺化遙測資料。這些元件使用來自 核發射器元件的遙測資料。如需詳細資訊，請參閱 [InfluxDB 發佈者元件](#) 的 README。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [相依性](#)
- [Configuration](#)
- [輸出資料](#)
- [用途](#)
- [本機日誌檔案](#)

- [變更記錄](#)

版本

此元件具有下列版本：

- 1.0.x

類型

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件 [發行版本的](#) 相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以在此 [AWS IoT Greengrass 主控台](#) 中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

1.0.12

下表列出此元件 1.0.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.17.0	軟式

1.0.11

下表列出此元件 1.0.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.16.0	硬式

1.0.10

下表列出此元件 1.0.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.15.0	硬式

1.0.9

下表列出此元件 1.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.14.0	硬式

1.0.8

下表列出此元件 1.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.13.0	硬式

1.0.7

下表列出此元件 1.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.12.0	硬式

1.0.6

下表列出此元件 1.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.11.0	硬式

1.0.5

下表列出此元件 1.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.10.0	硬式

1.0.4

下表列出此元件 1.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.9.0	硬式

1.0.3

下表列出此元件 1.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.8.0	硬式

1.0.2

下表列出此元件 1.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.7.0	硬式

1.0.1

下表列出此元件 1.0.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.6.0	硬式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.4.0 <2.5.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

pubSubPublish

(選用) 定義是否要將遙測資料發佈至`$local/greengrass/telemetry`主題。支援的值為 `true` 和 `false`。

預設：`true`

mqttTopic

(選用) 此元件發佈遙測資料的 AWS IoT Core MQTT 主題。

將此值設定為您要發佈遙測資料的 AWS IoT Core MQTT 主題。當此值為空時，核發射器不會將遙測資料發佈至 AWS 雲端。

 Note

使用 MQTT 主題將資料發佈到 AWS 雲端 需受[AWS IoT Core 定價](#)的限制。

預設：""

telemetryPublishIntervalMs

(選用) 元件發佈遙測資料的時間量 (以毫秒為單位)。如果您將此值設定為低於支援的最小值，則元件會改用最小值。

 Note

較低的發佈間隔會導致核心裝置上的 CPU 使用量提高。我們建議您從預設發佈間隔開始，並根據裝置的 CPU 用量進行調整。

下限：500

預設：60000

Example範例：組態合併更新

下列範例顯示範例組態合併更新，允許每 5 秒將遙測資料發佈至 \$local/greengrass/telemetry 主題和 greengrass/myTelemetry AWS IoT Core MQTT 主題。

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

輸出資料

此元件會將遙測指標發佈為下列主題的 JSON 陣列。

本機主題：\$local/greengrass/telemetry

您也可以選擇將遙測指標發佈至 AWS IoT Core MQTT 主題。如需主題的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT 主題](#)。

Example 範例資料

```
[
  {
    "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
}]
```

輸出陣列包含具有下列屬性的指標清單：

A

指標的彙總類型。

對於 CpuUsage 指標，此屬性設定為 `Average`，因為指標的發佈值是自上次發佈事件以來的平均 CPU 用量。

對於所有其他指標，核發射器不會彙總指標值，且此屬性設定為 `Count`。

N

指標的名稱

NS

指標命名空間。

TS

收集資料時的時間戳記。

U

指標值的單位。

V

指標值。

核發射器會發佈下列指標：

名稱	描述
系統	
SystemMemUsage	Greengrass 核心裝置上所有應用程式目前使用的記憶體數量，包括作業系統。
CpuUsage	Greengrass 核心裝置上所有應用程式目前使用的 CPU 數量，包括作業系統。
TotalNumberOfFDs	Greengrass 核心裝置的作業系統存放的檔案描述項數量。一個檔案描述項可唯一識別一個開啟的檔案。
Greengrass 核	
NumberOfComponentsRunning	在 Greengrass 核心裝置上執行的元件數量。
NumberOfComponentsErrored	Greengrass 核心裝置上處於錯誤狀態的元件數量。
NumberOfComponentsInstalled	安裝在 Greengrass 核心裝置上的元件數量。
NumberOfComponentsStarting	在 Greengrass 核心裝置上啟動的元件數量。
NumberOfComponentsNew	Greengrass 核心裝置上新增的元件數量。
NumberOfComponentsStopping	在 Greengrass 核心裝置上停止的元件數量。

名稱	描述	
NumberOfComponentsFinished	Greengrass 核心裝置上完成的元件數量。	
NumberOfComponentsBroken	Greengrass 核心裝置上損壞的元件數量。	
NumberOfComponentsStateless	Greengrass 核心裝置上無狀態的元件數量。	

用途

若要使用系統運作狀態遙測資料，您可以建立自訂元件，以訂閱核心發射器發佈遙測資料的主題，並視需要對該資料做出反應。由於核發射器元件提供將遙測資料發佈至本機主題的選項，因此您可以訂閱該主題，並使用已發佈的資料在核心裝置上於本機採取行動。然後，即使核心裝置與雲端的連線能力有限，也可以對遙測資料做出反應。

例如，您可以設定監聽遙測資料`$local/greengrass/telemetry`主題的元件，並將資料傳送至串流管理員元件，以將資料串流至 AWS 雲端。如需建立此類元件的詳細資訊，請參閱 [發佈/訂閱本機訊息](#) 和 [建立使用串流管理員的自訂元件](#)。

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.0.12	針對 Greengrass nucleus 2.16.0 版更新版本。
1.0.11	針對 Greengrass nucleus 2.15.0 版更新版本。
1.0.10	針對 Greengrass nucleus 2.14.0 版更新版本。
1.0.9	針對 Greengrass nucleus 2.13.0 版更新版本。
1.0.8	針對 Greengrass nucleus 2.12.0 版更新版本。
1.0.7	針對 Greengrass nucleus 2.11.0 版更新版本。
1.0.6	針對 Greengrass nucleus 2.10.0 版更新版本。
1.0.5	針對 Greengrass nucleus 2.9.0 版更新版本。
1.0.4	針對 Greengrass nucleus 2.8.0 版更新版本。
1.0.3	針對 Greengrass nucleus 2.7.0 版更新版本。
1.0.2	針對 Greengrass nucleus 2.6.0 版更新版本。
1.0.1	針對 Greengrass nucleus 2.5.0 版更新版本。
1.0.0	初始版本。

PKCS#11 供應商

PKCS#11 提供者元件 (`aws.greengrass.crypto.Pkcs11Provider`) 可讓您設定 AWS IoT Greengrass Core 軟體，以透過 [PKCS#11 介面](#) 使用硬體安全模組 (HSM)。此元件可讓您安全地存放憑證和私有金鑰檔案，使其不會在軟體中公開或複製。如需詳細資訊，請參閱[硬體安全整合](#)。

若要佈建將其憑證和私有金鑰存放在 HSM 中的 Greengrass 核心裝置，您必須在安裝 AWS IoT Greengrass Core 軟體時將此元件指定為佈建外掛程式。如需詳細資訊，請參閱[使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

AWS IoT Greengrass 提供此元件做為 JAR 檔案，您可以在安裝期間下載此檔案以指定做為佈建外掛程式。您可以將元件 JAR 檔案的最新版本下載為下列 URL：<https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.0.x

類型

此元件是外掛程式元件 (`aws.greengrass.plugin`)。[Greengrass 核心](#)會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 支援 [PKCS#1 v1.5](#) 簽章結構描述和具有 RSA-2048 金鑰大小（或更大）或 ECC 金鑰的 RSA 金鑰的硬體安全模組。

Note

若要搭配 ECC 金鑰使用硬體安全模組，您必須使用 [Greengrass nucleus](#) v2.5.6 或更新版本。

若要使用硬體安全模組和[秘密管理員](#)，您必須搭配 RSA 金鑰使用硬體安全模組。

- AWS IoT Greengrass 核心軟體可在執行時間（使用 libdl）載入的 PKCS#11 提供者程式庫，以叫用 PKCS#11 函數。PKCS#11 提供者程式庫必須實作下列 PKCS#11 API 操作：

- 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
- 硬體模組必須可透過插槽標籤根據 PKCS#11 規格中的定義來解析。
- 您必須將私有金鑰和憑證存放在相同插槽的 HSM 中，如果 HSM 支援物件 ID，則必須使用相同的物件標籤和物件 IDs。
- 憑證和私有金鑰必須透過物件標籤解析。
- 私有金鑰必須具有下列許可：
 - sign
 - decrypt
- (選用) 若要使用[秘密管理員元件](#)，您必須使用 2.1.0 版或更新版本，且私有金鑰必須具有下列許可：
 - unwrap
 - wrap
- (選用) 如果您使用 TPM2 程式庫並執行 Greengrass 核心做為服務，則必須提供具有 PKCS#11 存放區位置的環境變數。下列範例是具有必要環境變數的系統化服務檔案：

```
[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
```

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.0.11

下表列出此元件 2.0.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.17.0	軟式

2.0.10

下表列出此元件 2.0.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.16.0	軟式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.15.0	軟式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.14.0	軟式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.13.0	軟式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.12.0	軟式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.11.0	軟式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.10.0	軟式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.9.0	軟式

2.0.2

下表列出此元件 2.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.8.0	軟式

2.0.1

下表列出此元件 2.0.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.7.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.3 <2.6.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

name

PKCS#11 組態的名稱。

library

AWS IoT Greengrass 核心軟體可以使用 libdl 載入之 PKCS#11 實作程式庫的絕對檔案路徑。

slot

包含私有金鑰和裝置憑證的插槽 ID。此值與槽索引或槽標籤不同。

userPin

用來存取插槽的使用者 PIN 碼。

Example範例：組態合併更新

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.0.11	針對 Greengrass nucleus 2.16.0 版更新版本。
2.0.10	針對 Greengrass nucleus 2.15.0 版更新版本。
2.0.9	針對 Greengrass nucleus 2.14.0 版更新版本。
2.0.8	針對 Greengrass nucleus 2.13.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.12.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.11.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.10.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.9.0 版更新版本。

版本	變更
2.0.3	針對 Greengrass nucleus 2.8.0 版更新版本。
2.0.2	針對 Greengrass nucleus 2.7.0 版更新版本。
2.0.1	針對 Greengrass nucleus 2.6.0 版更新版本。
2.0.0	初始版本。

秘密管理員

秘密管理員元件 (`aws.greengrass.SecretManager`) 會將秘密從 AWS Secrets Manager 部署到 Greengrass 核心裝置。使用此元件可在 Greengrass 核心裝置上的自訂元件中安全地使用登入資料，例如密碼。如需 Secrets Manager 的詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的 [什麼是 AWS Secrets Manager？](#)。

若要在自訂 Greengrass 元件中存取此元件的秘密，請使用 [GetSecretValue](#) 操作 AWS IoT Device SDK。如需詳細資訊，請參閱 [使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#) 及 [擷取秘密值](#)。

此元件會加密核心裝置上的秘密，以保護您的登入資料和密碼，直到您需要使用為止。它使用核心裝置的私有金鑰來加密和解密秘密。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核心會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- [Greengrass 裝置角色](#) 必須允許 secretsmanager:GetSecretValue 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ]
    }
  ]
}
```

```
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:MySecret"
    ]
}
]
```

Note

如果您使用客戶受管 AWS Key Management Service 金鑰來加密秘密，裝置角色也必須允許 `kms:Decrypt` 動作。

如需 Secrets Manager IAM 政策的詳細資訊，請參閱 AWS Secrets Manager 《使用者指南》中的下列內容：

- [的身分驗證和存取控制 AWS Secrets Manager](#)
- [您可以在 IAM 政策或 的秘密政策中使用的動作、資源和內容金鑰 AWS Secrets Manager](#)
- 自訂元件必須定義授權政策，`aws:greengrass:GetSecretValue` 允許 取得您與此元件一起存放的秘密。在此授權政策中，您可以限制元件對特定秘密的存取。如需詳細資訊，請參閱 [秘密管理員 IPC 授權](#)。
- (選用) 如果您將核心裝置的私有金鑰和憑證存放在 [硬體安全模組](#) (HSM) 中，HSM 必須支援 RSA 金鑰，私有金鑰必須具有 `unwrap` 許可，而公有金鑰必須具有 `wrap` 許可。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱 [允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
secretsmanager. <code>region</code> .amazonaws.com	443	是	將秘密下載至核心裝置。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.2.7

下表列出此元件 2.2.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.17.0	軟式

2.2.6

下表列出此元件 2.2.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.16.0	軟式

2.2.2 – 2.2.5

下表列出此元件 2.2.2 到 2.2.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.15.0	軟式

2.2.0

下表列出此元件 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.13.0 <2.14.0	軟式

2.1.7 – 2.1.8

下表列出此元件 2.1.7 和 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.13.0	軟式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.12.0	軟式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.11.0	軟式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.10.0	軟式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.9.0	軟式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.8.0	軟式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.7.0	軟式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.6.0	軟式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

2.0.4 and 2.0.5

下表列出此元件 2.0.4 和 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

periodicRefreshIntervalMin (選用)

此元件會將核心裝置上的已設定秘密與 AWS Secrets Manager 來自服務的更新秘密值同步的間隔，以分鐘為單位。如果未設定此間隔，秘密管理員不會定期重新整理設定的秘密。

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef"
    }
  ],
  "periodicRefreshIntervalMin" : 60
}
```

cloudSecrets

要部署至核心裝置的 Secrets Manager 秘密清單。您可以指定標籤來定義要部署的每個秘密版本。如果您未指定版本，此元件會部署AWSCURRENT附加預備標籤的版本。如需詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[預備標籤](#)。

秘密管理員元件會在本機快取秘密。如果秘密值在 Secrets Manager 中變更，此元件不會自動擷取新值。若要更新本機複本，請將新標籤提供給秘密，並設定此元件以擷取由新標籤識別的 secret。

每個物件都包含下列資訊：

arn

要部署之秘密的 ARN。秘密的 ARN 可以是完整 ARN 或部分 ARN。我們建議您指定完整的 ARN，而不是部分 ARN。如需詳細資訊，請參閱[從部分 ARN 尋找秘密](#)。以下是完整 ARN 和部分 ARN 的範例：

- 完整 ARN：arn:aws:secretsmanager:us-east-2:111122223333:secret:*SecretName*-abcdef
- 部分 ARN：arn:aws:secretsmanager:us-east-2:111122223333:secret:*SecretName*

labels

(選用) 標籤清單，用於識別要部署至核心裝置的秘密版本。

每個標籤必須是字串。

Example範例：組態合併更新

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    }
  ]
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.2.7	針對 Greengrass nucleus 2.16.0 版更新版本。
2.2.6	錯誤修正和改善 修正秘密管理員因為緩慢或沒有回應的信任平台模組而無法取得秘密的問題。 針對 Greengrass nucleus 2.15.0 版更新版本。
2.2.5	錯誤修正和改善 修正如果本機快取中不存在秘密 AWS 雲端，則不會從擷取秘密的問題。
2.2.4	錯誤修正和改善 降低寫入本機秘密存放區的頻率。Secret Manager 現在只會在秘密更新時寫入本機存放區。
2.2.3	錯誤修正和改善 修正以下問題：當核心裝置離線且裝置的安全服務（例如 HSM）無法使用時，秘密管理員會抹除本機保留的秘密。
2.2.2	錯誤修正和改善 修正秘密管理員未下載設定部分 arn 之秘密的問題。
2.2.1	錯誤修正和改善 在 2.5.0 版及更高版本上支援秘密管理員。

版本	變更
2.2.0	新功能 新增透過新元件組態金鑰定期重新整理已設定秘密的支援。 在 GetSecretValue IPC 請求中新增對新請求參數的支援，以重新整理每個請求的秘密
2.1.8	錯誤修正和改善 修正秘密管理員不接受部分 arn 的問題。
2.1.7	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.5	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.4	錯誤修正和改善 修正部署秘密管理員並重新啟動 Greengrass 核時，快取秘密遭到移除的問題。 針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.0	新功能 - 新增硬體安全整合的支援。秘密管理員元件可以使用您存放在硬體安全模組 (HSM) 中的私有金鑰來加密和解密秘密。如需詳細資訊，請參閱硬體安全整合。 錯誤修正和改善 - 針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.9	針對 Greengrass nucleus 2.4.0 版更新版本。

版本	變更
2.0.8	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.5	改進 • 新增對 AWS 中國區域和 AWS GovCloud (US) 區域的支援。
2.0.4	初始版本。

安全通道

透過 `aws.greengrass.SecureTunneling` 元件，您可以與位於受限防火牆後方的 Greengrass 核心裝置建立安全的雙向通訊。

例如，假設您在防火牆後方有一個 Greengrass 核心裝置，禁止所有傳入連線。安全通道使用 MQTT 將存取字符傳輸至裝置，然後使用 WebSockets 透過防火牆與裝置進行 SSH 連線。使用此 AWS IoT 受管通道，您可以開啟裝置所需的 SSH 連線。如需使用 AWS IoT 安全通道連線至遠端裝置的詳細資訊，請參閱《AWS IoT 開發人員指南》中的[AWS IoT 安全通道](#)。

此元件會訂閱 `$aws/things/greengrass-core-device/tunnels/notify` 主題上的 AWS IoT Core MQTT 訊息中介裝置，以接收安全通道通知。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [授權](#)
- [用途](#)

- [另請參閱](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 1.1.x
- 1.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

架構：

- Armv71
- Armv8 (AArch64)
- x86_64

要求

此元件有下列需求：

- 安全通道元件可用的磁碟空間至少為 32 MB。此要求不包含 Greengrass 核心軟體或在相同裝置上執行的其他元件。
- 安全通道元件至少可使用 16 MB RAM。此要求不包含 Greengrass 核心軟體或在相同裝置上執行的其他元件。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。
- 安全通道元件 1.0.12 版和更新版本需要 GNU C Library (glibc) 2.25 版或更高版本，Linux 核心為 3.2 版或更高版本。不支援超過長期支援生命週期結束日期的作業系統和程式庫版本。您應該使用具有長期支援的作業系統和程式庫。

- 作業系統和 Java 執行期都必須安裝為 64 位元。
- 在 Greengrass 核心裝置上安裝 [Python](#) 3.5 或更新版本，並新增至 PATH 環境變數。
- libcrypto.so.1.1 安裝在 Greengrass 核心裝置上，並新增至 PATH 環境變數。
- 在 Greengrass 核心裝置上的連接埠 443 上開啟傳出流量。
- 開啟您要用來與 Greengrass 核心裝置通訊的通訊服務支援。例如，若要開啟與裝置的 SSH 連線，您必須在該裝置上開啟 SSH。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	連線埠	必要	說明
data.tunneling.iot . <i>region</i> .amazonaws.com	443	是	建立安全通道。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

1.0.19 – 1.1.3

下表列出此元件 1.0.19 到 1.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式

1.0.18

下表列出此元件 1.0.18 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式

1.0.16 – 1.0.17

下表列出此元件 1.0.16 至 1.0.17 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式

1.0.14 – 1.0.15

下表列出此元件 1.0.14 至 1.0.15 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式

1.0.11 – 1.0.13

下表列出此元件 1.0.11 – 1.0.13 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式

1.0.10

下表列出此元件 1.0.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式

1.0.9

下表列出此元件 1.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式

1.0.8

下表列出此元件 1.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

1.0.5 - 1.0.7

下表列出此元件 1.0.5 到 1.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式

1.0.4

下表列出此元件 1.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式

1.0.3

下表列出此元件 1.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式

1.0.2

下表列出此元件 1.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式

1.0.1

下表列出此元件 1.0.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式

1.0.0

下表列出此元件 1.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

OS_DIST_INFO

(選用) 核心裝置的作業系統。根據預設，元件會嘗試自動識別核心裝置上執行的作業系統。如果元件無法以預設值開頭，請使用此值來指定作業系統。如需此元件支援的作業系統清單，請參閱 [裝置要求](#)。

此值可以是下列其中一項：auto、ubuntu、amzn2、raspberrypi。

預設：auto

accessControl

(選用) 包含[授權政策](#)的物件，允許元件訂閱安全通道通知主題。

 Note

如果您的部署以物件群組為目標，請勿修改此組態參數。如果您的部署以個別核心裝置為目標，而且您想要將其訂閱限制為裝置的主題，請指定核心裝置的物件名稱。在裝置的授權政策中的 resources 值中，將 MQTT 主題萬用字元取代為裝置的物件名稱。

```
{
  "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範例：組態合併更新

下列範例組態指定，允許此元件在**MyGreengrassCore**執行 Ubuntu 的核心裝置上開啟安全通道。

```
{
```

```
"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"
      ]
    }
  }
}
```

本機日誌檔案

此元件使用以下日誌檔案。

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

授權

此元件包含下列第三方軟體/授權：

- [AWS IoT 裝置用戶端](#)/Apache 授權 2.0
- [適用於 JAVA 的 AWS IoT Device SDK](#)/Apache License 2.0
- [gson](#)/Apache 授權 2.0
- [log4j](#)/Apache 授權 2.0
- [slf4j](#)/Apache 授權 2.0

用途

若要在裝置上使用安全通道元件，請執行下列動作：

1. 將安全通道元件部署至您的裝置。
2. 開啟 [AWS IoT 主控台](#)。從左側功能表中，選擇遠端動作，然後選擇安全通道。
3. 為您的 Greengrass 裝置建立通道。
4. 下載來源存取字符。
5. 使用本機代理搭配來源存取字符來連線至目的地。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[如何使用本機代理](#)。

另請參閱

- 《AWS IoT 開發人員指南》中的[AWS IoT 安全通道](#)
- 《AWS IoT 開發人員指南》中的[如何使用本機代理](#)

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.1.3	錯誤修正和改善 • 將元件調用的基礎AWS IoT 裝置用戶端從 1.10.0 版升級至 1.10.1 版。 • 修正安全通道元件 1.1.2 版中的 GNU C 程式庫 (glibc) 相容性問題。
1.1.2	 Warning 此版本不再可用。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 • 將元件叫用的基礎AWS IoT 裝置用戶端從 1.9.0 版升級至 1.10.0 版。 • 修正承載傳輸問題，以防止使用者透過安全通道將大型檔案從 Greengrass V2 核心裝置轉送至來源裝置。

版本	變更
1.1.1	錯誤修正和改善 • 新增支援 Greengrass nucleus lite 的組態。
1.1.0	新功能 • 新增 Greengrass nucleus lite 的配方支援。
1.0.19	錯誤修正和改善 • 將元件調用的基礎AWS IoT 裝置用戶端從 1.8.0 版升級至 1.9.0 版。 • 將元件層級的並行通道限制增加到 20 個通道。 • 將預設 AWS IoT Greengrass 核心 IPC 逾時從 3 秒增加到 10 秒。  Warning 如果您使用安全通道本機代理做為通道來源用戶端，請勿將元件更新至 此版本，除非您也已將本機代理升級至 3.1.1 版或更新版本。
1.0.18	針對 Greengrass nucleus 2.12.0 版更新版本。
1.0.17	錯誤修正和改善 • 修正阻止使用者建立通道的執行緒清除問題。此元件現在會在收到 CloseTunnel 訊號後或通道在 12 小時後過期時清除執行緒。
1.0.16	針對 Greengrass nucleus 2.11.0 版更新版本。
1.0.15	錯誤修正和改善 • 修正裝置上沒有主目錄的使用者啟動問題。安全通道元件現在無需為影子文件建立目錄即可啟動。
1.0.14	針對 Greengrass nucleus 2.10.0 版更新版本。
1.0.13	錯誤修正和改善 • 修正孤立用戶端程序防止多個通道以裝置為目標的問題。

版本	變更
1.0.12	錯誤修正和改善 在 Raspberry Pi 作業系統上執行時，新增對 x86_64 (AMD64) 和 ARMv8 (Aarch64) 的支援。
1.0.11	針對 Greengrass nucleus 2.9.0 版更新版本。
1.0.10	針對 Greengrass nucleus 2.8.0 版更新版本。
1.0.9	針對 Greengrass nucleus 2.7.0 版更新版本。
1.0.8	針對 Greengrass nucleus 2.6.0 版更新版本。
1.0.7	錯誤修正和改善 修正當您透過 SCP 傳輸大型檔案時，元件中斷連線的問題。
1.0.6	此版本包含錯誤修正。
1.0.5	針對 Greengrass nucleus 2.5.0 版更新版本。
1.0.4	針對 Greengrass nucleus 2.4.0 版更新版本。
1.0.3	針對 Greengrass nucleus 2.3.0 版更新版本。
1.0.2	針對 Greengrass nucleus 2.2.0 版更新版本。
1.0.1	針對 Greengrass nucleus 2.1.0 版更新版本。
1.0.0	初始版本。

Shadow 管理員

影子管理員元件 (aws.greengrass.ShadowManager) 會在您的核心裝置上啟用本機影子服務。本機影子服務允許元件使用程序間通訊與[本機影子互動](#)。影子管理員元件會管理本機影子文件的儲存，也會處理本機影子狀態與 AWS IoT Device Shadow 服務的同步處理。

如需 Greengrass 核心裝置如何與影子互動的詳細資訊，請參閱 [與裝置陰影互動](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是外掛程式元件 (aws.greengrass.plugin)。 [Greengrass 核心](#) 會在與核心相同的 Java 虛擬機器 (JVM) 中執行此元件。當您在核心裝置上變更此元件的版本時，核會重新啟動。

此元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- （選用）若要將影子同步至 AWS IoT Device Shadow 服務，Greengrass 核心裝置 AWS IoT 的政策必須允許下列 AWS IoT Core 影子政策動作：
 - `iot:GetThingShadow`
 - `iot:UpdateThingShadow`
 - `iot:DeleteThingShadow`

如需這些 AWS IoT Core 政策的詳細資訊，請參閱《AWS IoT 開發人員指南》中的[AWS IoT Core 政策動作](#)。

如需最低 AWS IoT 政策的詳細資訊，請參閱 [AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)

- 影子管理員元件支援在 VPC 中執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

2.3.13

下表列出此元件 2.3.13 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	<code>>=2.5.0 <2.17.0</code>	軟式

2.3.12

下表列出此元件 2.3.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.17.0	軟式

2.3.11

下表列出此元件 2.3.11 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.16.0	軟式

2.3.10

下表列出此元件 2.3.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.15.0	軟式

2.3.9

下表列出此元件 2.3.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.14.0	軟式

2.3.5 – 2.3.8

下表列出此元件 2.3.5 到 2.3.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.13.0	軟式

2.3.3 and 2.3.4

下表列出此元件 2.3.3 和 2.3.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.12.0	軟式

2.3.2

下表列出此元件 2.3.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.11.0	軟式

2.3.0 and 2.3.1

下表列出此元件 2.3.0 和 2.3.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.5.0 <2.10.0	軟式

2.2.3 and 2.2.4

下表列出此元件 2.2.3 和 2.2.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <3.0.0	軟式

2.2.2

下表列出此元件 2.2.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.9.0	軟式

2.2.1

下表列出此元件 2.2.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.8.0	軟式

2.1.1 and 2.2.0

下表列出此元件 2.1.1 和 2.2.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.7.0	軟式

2.0.5 - 2.1.0

下表列出此元件 2.0.5 到 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.6.0	軟式

2.0.3 and 2.0.4

下表列出此元件 2.0.3 和 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.5.0	軟式

2.0.1 and 2.0.2

下表列出此元件 2.0.1 和 2.0.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.4.0	軟式

2.0.0

下表列出此元件 2.0.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.2.0 <2.3.0	軟式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.3.x

strategy

(選用) 此元件用來在 AWS IoT Core 和核心裝置之間同步陰影的策略。

此物件包含下列資訊。

type

(選用) 此元件用來在 AWS IoT Core 和 核心裝置之間同步陰影的策略類型。您可以從以下選項中選擇：

- `realTime` – AWS IoT Core 每次進行陰影更新時，同步陰影。
- `periodic` – AWS IoT Core 以您使用`delay`組態參數指定的定期間隔將陰影與 同步。

預設：`realTime`

delay

(選用) 當您指定同步策略時 AWS IoT Core，此元件與 periodic 同步陰影的間隔，以秒為單位。

Note

如果您指定 periodic 同步策略，則需要此參數。

synchronize

(選用) 決定影子如何與 同步的同步設定 AWS 雲端。

Note

您必須使用此屬性建立組態更新，才能與 同步陰影 AWS 雲端。

此物件包含下列資訊。

coreThing

(選用) 要同步的核心裝置陰影。此物件包含下列資訊。

classic

(選用) 根據預設，影子管理員會使用 同步核心裝置傳統影子的本機狀態 AWS 雲端。如果您不想同步傳統裝置影子，請將此設為 false。

預設：true

namedShadows

(選用) 要同步的具名核心裝置影子清單。您必須指定影子的確切名稱。

Warning

AWS IoT Greengrass 服務使用 AWSManagedGreengrassV2Deployment 具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

shadowDocumentsMap

(選用) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定影子文件。我們建議您使用此參數，而非 `shadowDocuments` 物件。

Note

如果您指定 `shadowDocumentsMap` 物件，則不得指定 `shadowDocuments` 物件。

每個物件都包含下列資訊：

thingName

此影子組態之 *thingName* 的影子組態。

`classic`

(選用) 如果您不想同步 `thingName` 裝置的傳統裝置影子，請將此設為 `false`。

`namedShadows`

您要同步的具名影子清單。您必須指定影子的確切名稱。

`shadowDocuments`

(選用) 要同步的其他裝置陰影清單。我們建議您改用 `shadowDocumentsMap` 參數。

Note

如果您指定 `shadowDocuments` 物件，則不得指定 `shadowDocumentsMap` 物件。

此清單中的每個物件都包含下列資訊。

`thingName`

要同步陰影之裝置的物件名稱。

`classic`

(選用) 如果您不想同步 `thingName` 裝置的傳統裝置影子，請將此設為 `false`。

預設：`true`

namedShadows

(選用) 您要同步的具名裝置陰影清單。您必須指定影子的確切名稱。

direction

(選用) 在本機影子服務與 之間同步影子的方向 AWS 雲端。您可以設定此選項，以減少頻寬和對 的連線 AWS 雲端。您可以從以下選項中選擇：

- `betweenDeviceAndCloud` – 同步本機影子服務與 之間的影子 AWS 雲端。
- `deviceToCloud` – 將陰影更新從本機陰影服務傳送至 AWS 雲端，並忽略來自 的陰影更新 AWS 雲端。
- `cloudToDevice` – 從 接收陰影更新 AWS 雲端，而且不要從本機陰影服務傳送陰影更新至 AWS 雲端。

預設：BETWEEN_DEVICE_AND_CLOUD

rateLimits

(選用) 決定陰影服務請求速率限制的設定。

此物件包含下列資訊。

maxOutboundSyncUpdatesPerSecond

(選用) 裝置每秒傳輸的同步請求數目上限。

預設：100 個請求/秒

maxTotalLocalRequestsRate

(選用) 每秒傳送至核心裝置的本機 IPC 請求數量上限。

預設：200 個請求/秒

maxLocalRequestsPerSecondPerThing

(選用) 每秒針對每個連線 IoT 物件傳送的本機 IPC 請求數目上限。

預設：每個物件每秒 20 個請求

Note

這些速率限制參數會定義本機影子服務的每秒請求數上限。AWS IoT Device Shadow 服務的每秒請求數上限取決於您的 AWS 區域。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制Amazon Web Services 一般參考。

shadowDocumentSizeLimitBytes

(選用) 本機影子的每個 JSON 狀態文件允許的大小上限。

如果您增加此值，您還必須增加雲端影子的 JSON 狀態文件的資源限制。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制Amazon Web Services 一般參考。

預設：8192 位元組

上限：30720 位元組

Example範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含影子管理員元件的所有可用組態參數。

```
{
  "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

(選用) 此元件用來在 AWS IoT Core 和核心裝置之間同步陰影的策略。

此物件包含下列資訊。

type

(選用) 此元件用來在 AWS IoT Core 和核心裝置之間同步陰影的策略類型。您可以從以下選項中選擇：

- `realTime` – AWS IoT Core 每次進行陰影更新時，同步陰影。
- `periodic` – AWS IoT Core 以您使用 `delay` 組態參數指定的定期間隔，將陰影與 同步。

預設：`realTime`

delay

(選用) 當您指定同步策略時 AWS IoT Core，此元件與 `periodic` 同步陰影的間隔，以秒為單位。

Note

如果您指定 `periodic` 同步策略，則需要此參數。

synchronize

(選用) 決定影子如何與 同步的同步設定 AWS 雲端。

Note

您必須使用此屬性建立組態更新，才能將影子與 同步 AWS 雲端。

此物件包含下列資訊。

coreThing

(選用) 要同步的核心裝置陰影。此物件包含下列資訊。

classic

(選用) 根據預設，影子管理員會使用 同步核心裝置傳統影子的本機狀態 AWS 雲端。如果您不想同步傳統裝置影子，請將此設為 `false`。

預設：`true`

namedShadows

(選用) 要同步的具名核心裝置陰影清單。您必須指定影子的確切名稱。

Warning

AWS IoT Greengrass 服務使用 `AWSThingsShadowV2Deployment` 具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

shadowDocumentsMap

(選用) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定影子文件。我們建議您使用此參數，而非 `shadowDocuments` 物件。

Note

如果您指定 `shadowDocumentsMap` 物件，則不得指定 `shadowDocuments` 物件。

每個物件都包含下列資訊：

thingName

此影子組態之 *thingName* 的影子組態。

classic

(選用) 如果您不想同步 *thingName* 裝置的傳統裝置陰影，請將此設為 `false`。

namedShadows

您要同步的具名影子清單。您必須指定影子的確切名稱。

shadowDocuments

(選用) 要同步的其他裝置陰影清單。我們建議您改用 `shadowDocumentsMap` 參數。

 Note

如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

thingName

要同步陰影之裝置的物件名稱。

classic

(選用) 如果您不想同步thingName裝置的傳統裝置陰影，請將此設為 false。

預設：true

namedShadows

(選用) 您要同步的具名裝置陰影清單。您必須指定影子的確切名稱。

direction

(選用) 在本機影子服務與 之間同步影子的方向 AWS 雲端。您可以設定此選項，以減少頻寬和對 的連線 AWS 雲端。您可以從以下選項中選擇：

- betweenDeviceAndCloud – 同步本機影子服務與 之間的影子 AWS 雲端。
- deviceToCloud – 將陰影更新從本機陰影服務傳送至 AWS 雲端，並忽略來自 的陰影更新 AWS 雲端。
- cloudToDevice – 從 接收陰影更新 AWS 雲端，而且不要將陰影更新從本機陰影服務傳送至 AWS 雲端。

預設：BETWEEN_DEVICE_AND_CLOUD

rateLimits

(選用) 決定陰影服務請求速率限制的設定。

此物件包含下列資訊。

maxOutboundSyncUpdatesPerSecond

(選用) 裝置每秒傳輸的同步請求數目上限。

預設：100 個請求/秒

maxTotalLocalRequestsRate

(選用) 每秒傳送至核心裝置的本機 IPC 請求數目上限。

預設：200 個請求/秒

maxLocalRequestsPerSecondPerThing

(選用) 每秒針對每個連線 IoT 物件傳送的本機 IPC 請求數目上限。

預設：每個物件每秒 20 個請求

Note

這些速率限制參數會定義本機影子服務的每秒請求數上限。AWS IoT Device Shadow 服務的每秒請求數上限取決於您的 AWS 區域。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制Amazon Web Services 一般參考。

shadowDocumentSizeLimitBytes

(選用) 本機影子的每個 JSON 狀態文件允許的大小上限。

如果您增加此值，您還必須增加雲端影子的 JSON 狀態文件的資源限制。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制Amazon Web Services 一般參考。

預設：8192 位元組

上限：30720 位元組

Example範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含影子管理員元件的所有可用組態參數。

```
{
  "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

(選用) 此元件用來在 AWS IoT Core 和核心裝置之間同步陰影的策略。

此物件包含下列資訊。

type

(選用) 此元件用來在 AWS IoT Core 和核心裝置之間同步陰影的策略類型。您可以從以下選項中選擇：

- **realTime** – AWS IoT Core 每次進行陰影更新時，同步陰影。
- **periodic** – AWS IoT Core 以您使用 **delay** 組態參數指定的定期間隔，將陰影與 同步。

預設：**realTime**

delay

(選用) 當您指定同步策略時 AWS IoT Core，此元件與 **periodic** 同步陰影的間隔，以秒為單位。

 Note

如果您指定 `periodic` 同步策略，則需要此參數。

synchronize

(選用) 決定影子如何與 同步的同步設定 AWS 雲端。

 Note

您必須使用此屬性建立組態更新，才能將影子與 同步 AWS 雲端。

此物件包含下列資訊。

coreThing

(選用) 要同步的核心裝置陰影。此物件包含下列資訊。

classic

(選用) 根據預設，影子管理員會使用 同步核心裝置傳統影子的本機狀態 AWS 雲端。如果您不想同步傳統裝置影子，請將此設為 `false`。

預設：`true`

namedShadows

(選用) 要同步的具名核心裝置陰影清單。您必須指定影子的確切名稱。

 Warning

AWS IoT Greengrass 服務使用 `AWSThingsV2Deployment` 具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

shadowDocumentsMap

(選用) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定影子文件。我們建議您使用此參數，而非 `shadowDocuments` 物件。

 Note

如果您指定shadowDocumentsMap物件，則不得指定shadowDocuments物件。

每個物件都包含下列資訊：

thingName

此影子組態之 *thingName* 的影子組態。

classic

(選用) 如果您不想同步thingName裝置的傳統裝置陰影，請將此設為 false。

namedShadows

您要同步的具名影子清單。您必須指定影子的確切名稱。

shadowDocuments

(選用) 要同步的其他裝置陰影清單。我們建議您改用 shadowDocumentsMap 參數。

 Note

如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

thingName

要同步陰影之裝置的物件名稱。

classic

(選用) 如果您不想同步thingName裝置的傳統裝置陰影，請將此設為 false。

預設：true

namedShadows

(選用) 您要同步的具名裝置陰影清單。您必須指定影子的確切名稱。

rateLimits

(選用) 決定陰影服務請求速率限制的設定。

此物件包含下列資訊。

`maxOutboundSyncUpdatesPerSecond`

(選用) 裝置每秒傳輸的同步請求數目上限。

預設：100 個請求/秒

`maxTotalLocalRequestsRate`

(選用) 每秒傳送至核心裝置的本機 IPC 請求數量上限。

預設：200 個請求/秒

`maxLocalRequestsPerSecondPerThing`

(選用) 每秒針對每個連線 IoT 物件傳送的本機 IPC 請求數目上限。

預設：每個物件每秒 20 個請求

 Note

這些速率限制參數會定義本機影子服務的每秒請求數上限。AWS IoT Device Shadow 服務的每秒請求數上限取決於您的 AWS 區域。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制 Amazon Web Services 一般參考。

`shadowDocumentSizeLimitBytes`

(選用) 本機影子的每個 JSON 狀態文件允許的大小上限。

如果您增加此值，您還必須增加雲端影子的 JSON 狀態文件的資源限制。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制 Amazon Web Services 一般參考。

預設：8192 位元組

上限：30720 位元組

Example 範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含影子管理員元件的所有可用組態參數。

```
{  
  "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

(選用) 決定影子如何與 同步的同步設定 AWS 雲端。

Note

您必須使用此屬性建立組態更新，才能與 同步陰影 AWS 雲端。

此物件包含下列資訊。

coreThing

(選用) 要同步的核心裝置陰影。此物件包含下列資訊。

classic

(選用) 根據預設，影子管理員會使用 同步核心裝置傳統影子的本機狀態 AWS 雲端。如果您不想同步傳統裝置影子，請將此設為 `false`。

預設：`true`

namedShadows

(選用) 要同步的具名核心裝置影子清單。您必須指定影子的確切名稱。

Warning

AWS IoT Greengrass 服務使用 `AWSThingsShadow` 具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

shadowDocumentsMap

(選用) 要同步的其他裝置陰影。使用此組態參數可讓您更輕鬆地指定影子文件。我們建議您使用此參數，而非 `shadowDocuments` 物件。

Note

如果您指定 `shadowDocumentsMap` 物件，則不得指定 `shadowDocuments` 物件。

每個物件都包含下列資訊：

thingName

此影子組態之 *thingName* 的影子組態。

classic

(選用) 如果您不想同步 *thingName* 裝置的傳統裝置影子，請將此設為 `false`。

namedShadows

您要同步的具名影子清單。您必須指定影子的確切名稱。

shadowDocuments

(選用) 要同步的其他裝置陰影清單。我們建議您改用 `shadowDocumentsMap` 參數。

 Note

如果您指定shadowDocuments物件，則不得指定shadowDocumentsMap物件。

此清單中的每個物件都包含下列資訊。

thingName

要同步陰影之裝置的物件名稱。

classic

(選用) 如果您不想同步thingName裝置的傳統裝置影子，請將此設為 false。

預設：true

namedShadows

(選用) 您要同步的具名裝置陰影清單。您必須指定影子的確切名稱。

rateLimits

(選用) 決定陰影服務請求速率限制的設定。

此物件包含下列資訊。

maxOutboundSyncUpdatesPerSecond

(選用) 裝置每秒傳輸的同步請求數目上限。

預設：100 個請求/秒

maxTotalLocalRequestsRate

(選用) 每秒傳送至核心裝置的本機 IPC 請求數量上限。

預設：200 個請求/秒

maxLocalRequestsPerSecondPerThing

(選用) 每秒針對每個連線 IoT 物件傳送的本機 IPC 請求數目上限。

預設：每個物件每秒 20 個請求

Note

這些速率限制參數會定義本機影子服務的每秒請求數上限。AWS IoT Device Shadow 服務的每秒請求數上限取決於您的 AWS 區域。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制Amazon Web Services 一般參考。

shadowDocumentSizeLimitBytes

(選用) 本機影子的每個 JSON 狀態文件允許的大小上限。

如果您增加此值，您還必須增加雲端影子的 JSON 狀態文件的資源限制。如需詳細資訊，請參閱《》中 [AWS IoT Device Shadow Service API](#) 的限制Amazon Web Services 一般參考。

預設：8192 位元組

上限：30720 位元組

Example範例：組態合併更新

下列範例顯示範例組態合併更新，其中包含影子管理員元件的所有可用組態參數。

```
{
  "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
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C:\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.3.13	錯誤修正和改善 - 修正即使同步方向組態允許，本機陰影更新也會間歇性地無法同步至雲端的問題。 - 少量错误修复和改进。
2.3.12	錯誤修正和改善 修正設定超過 1024 個裝置影子時，部署遭到封鎖的問題。
2.3.11	針對 Greengrass nucleus 2.15.0 版更新版本。
2.3.10	針對 Greengrass nucleus 2.14.0 版更新版本。
2.3.9	針對 Greengrass nucleus 2.13.0 版更新版本。
2.3.8	錯誤修正和改善 修正陰影管理員在 MQTT 用戶端連線期間建立死結情況的問題。
2.3.7	錯誤修正和改善 修正陰影管理員在陰影管理員同步期間定期記錄 <code>NullPointerException</code> 錯誤的問題。
2.3.6	錯誤修正和改善 修正重新取得連線後，裝置離線時透過 AWS 雲端 更新刪除的影子屬性繼續存在於本機影子中的問題。
2.3.5	針對 Greengrass nucleus 2.12.0 版更新版本。
2.3.4	錯誤修正和改善 新增對 null 和空白陰影狀態文件的支援。
2.3.3	針對 Greengrass nucleus 2.11.0 版更新版本。

版本	變更
2.3.2	錯誤修正和改善 • 修正當本機影子資料庫損毀時，影子管理員進入 BROKEN 狀態的問題。 • 針對 Greengrass nucleus 2.10.0 版更新版本。
2.3.1	錯誤修正和改善 • 修正可能導致雲端陰影更新無法同步的條件。 • 修正對具名影子同步組態的變更僅適用於一個具名影子的問題。
2.3.0	錯誤修正和改善 • 修正當 Greengrass 裝置私有金鑰存放在硬體安全模組中時，可能導致影子無法同步的問題。
2.2.4	錯誤修正和改善 • 修正更新本機影子文件時，影子大小驗證與雲端不一致的問題。 • 修正部署在組態節點RESET上執行時，陰影管理員會停止接聽組態更新的問題。
2.2.3	針對 Greengrass nucleus 2.9.0 版更新版本。
2.2.2	針對 Greengrass nucleus 2.8.0 版更新版本。
2.2.1	針對 Greengrass nucleus 2.7.0 版更新版本。

版本	變更
2.2.0	新功能 - 透過本機發佈/訂閱界面新增對本機影子服務的支援。您現在可以在影子 MQTT 主題上與本機發佈/訂閱訊息代理程式通訊，以取得、更新和刪除核心裝置上的影子。此功能可讓您使用 MQTT 橋接器，在用戶端裝置與本機發佈/訂閱界面之間轉送影子主題的訊息，將用戶端裝置連線至本機影子服務。 此功能需要 2.6.0 版或更新版本的 Greengrass 核元件。若要將用戶端裝置連線至本機影子服務，您還必須使用 MQTT 橋接器元件 的 v2.2.0 或更新版本。 - 新增您可以設定的 <code>direction</code> 選項，以自訂方向，在本機影子服務與之間同步影子 AWS 雲端。您可以設定此選項，以減少頻寬和對的連線 AWS 雲端。
2.1.1	錯誤修正和改善 - 修正 JSON 裝置陰影狀態文件 <code>desired</code> 和 <code>reported</code> 區段的最大深度為 4 個層級而非 5 個層級的問題。 - 針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.0	新功能 新增定期影子同步間隔的支援，因此您可以設定核心裝置以減少頻寬使用量和費用。
2.0.6	此版本包含錯誤修正和改善。
2.0.5	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.4	錯誤修正和改善 - 修正導致影子管理員刪除先前已刪除之任何影子新建立版本的問題。 - 更新 <code>DeleteThingShadow</code> IPC 操作，以在呼叫時增加陰影版本。
2.0.3	針對 Greengrass nucleus 2.4.0 版更新版本。

版本	變更
2.0.2	錯誤修正和改善 - 修正從 同步陰影狀態時，導致陰影管理員無法辨識 delta 屬性的問題 AWS IoT Core。 - 修正有時導致影子同步請求錯誤合併的問題。
2.0.1	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.0	初始版本。

Amazon SNS

Amazon SNS 元件 (aws.greengrass.SNS) 會將訊息發佈至 Amazon Simple Notification Service (Amazon SNS) 主題。您可以使用此元件，將事件從 Greengrass 核心裝置傳送至 Web 伺服器、電子郵件地址和其他訊息訂閱者。如需詳細資訊，請參閱《[Amazon Simple Notification Service 開發人員指南](#)》中的什麼是 [Amazon SNS](#)？。

若要使用此元件發佈至 Amazon SNS 主題，請將訊息發佈至此元件訂閱的主題。根據預設，此元件會訂閱sns/message[本機發佈/訂閱](#)主題。您可以在部署此元件時指定其他主題，包括 AWS IoT Core MQTT 主題。

在自訂元件中，您可能想要實作篩選或格式化邏輯，以便在將訊息發佈到此元件之前處理來自其他來源的訊息。這可讓您將訊息處理邏輯集中在單一元件上。

Note

此元件提供與 AWS IoT Greengrass V1 中的 Amazon SNS 連接器類似的功能。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的 [Amazon SNS 連接器](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)

- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [輸出資料](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是 Lambda 元件 (aws.greengrass.lambda)。 [Greengrass 核](#) 會使用 Lambda [啟動器元件執行此元件的 Lambda](#) 函數。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您希望核心裝置執行容器化 Lambda 函數，則裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 安裝在核心裝置上並新增至 PATH 環境變數的 [Python](#) 3.7 版。
- Amazon SNS 主題。如需詳細資訊，請參閱《Amazon Simple Notification Service 開發人員指南》中的[建立 Amazon SNS 主題](#)。

- [Greengrass 裝置角色](#) 必須允許 `sns:Publish` 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:us-east-1:123456789012:topic-name"
      ]
    }
  ]
}
```

您可以動態覆寫此元件輸入訊息承載中的預設主題。如果您的應用程式使用此功能，IAM 政策必須包含所有目標主題做為資源。您可以為資源授予精細或條件式存取 (例如，使用萬用字元 * 命名機制)。

- 若要從此元件接收輸出資料，您必須在部署此元件時合併[舊版訂閱路由器元件](#) (`aws.greengrass.LegacySubscriptionRouter`) 的下列組態更新。此組態會指定此元件發佈回應的主題。

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"
  }
}
```

- 以您使用 AWS 區域 的 取代##。
- 以此元件執行的 Lambda 函數版本取代 版本。若要尋找 Lambda 函數版本，您必須檢視要部署之此元件版本的配方。在[AWS IoT Greengrass 主控台](#)中開啟此元件的詳細資訊頁面，並尋找 Lambda 函數鍵/值對。此鍵/值對包含 Lambda 函數的名稱和版本。

Important

每次部署此元件時，都必須更新舊版訂閱路由器上的 Lambda 函數版本。這可確保針對您部署的元件版本使用正確的 Lambda 函數版本。

如需詳細資訊，請參閱[建立部署](#)。

- Amazon SNS 元件支援在 VPC 中執行。若要在 VPC 中部署此元件，需要下列項目。
 - Amazon SNS 元件必須具有sns.region.amazonaws.com具有 VPC 端點的連線com.amazonaws.us-east-1.sns能力。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
sns. <i>region</i> .amazonaws.com	443	是	將訊息發布到 Amazon SNS。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.10

下表列出此元件 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.16.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.9

下表列出此元件 2.1.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.15.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.8

下表列出此元件 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.14.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.7

下表列出此元件 2.1.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.6

下表列出此元件 2.1.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.1.5

下表列出此元件 2.1.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.4

下表列出此元件 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.3

下表列出此元件 2.1.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	^2.0.0	硬式

2.1.2

下表列出此元件 2.1.2 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.8 - 2.1.0

下表列出此元件 2.0.8 和 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字元交換服務	^2.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字元交換服務	^2.0.0	硬式

2.0.6

下表列出此元件 2.0.6 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.5

下表列出此元件 2.0.5 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
字符交換服務	^2.0.0	硬式

2.0.4

下表列出此元件 2.0.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	硬式
Lambda 啟動器	^2.0.0	硬式
Lambda 執行階段	^2.0.0	軟式
權杖交換服務	^2.0.0	硬式

2.0.3

下表列出此元件 2.0.3 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	硬式
Lambda 啟動器	>=1.0.0	硬式
Lambda 執行階段	>=1.0.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	>=1.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

Note

此元件的預設組態包含 Lambda 函數參數。建議您只編輯下列參數，以在裝置上設定此元件。

lambdaParams

包含此元件 Lambda 函數參數的物件。此物件包含下列資訊：

EnvironmentVariables

包含 Lambda 函數參數的物件。此物件包含下列資訊：

DEFAULT_SNS_ARN

此元件發佈訊息之預設 Amazon SNS 主題的 ARN。您可以使用輸入訊息承載中的 `sns_topic_arn` 屬性覆寫目的地主題。

containerMode

(選用) 此元件的容器化模式。您可以從以下選項中選擇：

- `NoContainer` – 元件不會在隔離的執行時間環境中執行。
- `GreengrassContainer` – 元件會在 AWS IoT Greengrass 容器內的隔離執行時間環境中執行。

預設：`GreengrassContainer`

containerParams

(選用) 包含此元件容器參數的物件。如果您 `GreengrassContainer` 為指定，元件會使用這些參數 `containerMode`。

此物件包含下列資訊：

memorySize

(選用) 要配置給元件的記憶體量 (以 KB 為單位)。

預設為 512 MB (525,312 KB)。

pubsubTopics

(選用) 物件，其中包含元件訂閱接收訊息的主題。您可以指定每個主題，以及元件是從 訂閱 MQTT 主題 AWS IoT Core 或本機發佈/訂閱主題。

此物件包含下列資訊：

0 – 這是做為字串的陣列索引。

包含下列資訊的物件：

type

(選用) 此元件用來訂閱訊息的發佈/訂閱訊息類型。您可以從以下選項中選擇：

- PUB_SUB - 訂閱本機發佈/訂閱訊息。如果您選擇此選項，主題不能包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
- IOT_CORE – 訂閱 AWS IoT Core MQTT 訊息。如果您選擇此選項，主題可以包含 MQTT 萬用字元。如需在指定此選項時如何從自訂元件傳送訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設：PUB_SUB

topic

(選用) 元件訂閱接收訊息的主題。如果您IotCore為 指定 type，則可以在本主題中使用 MQTT 萬用字元 (+ 和 #)。

Example範例：組態合併更新 (容器模式)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
```

```
}
```

Example範例：組態合併更新（無容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

輸入資料

此元件接受下列主題的訊息，並將訊息依原樣發佈至目標 Amazon SNS 主題。根據預設，此元件會訂閱本機發佈/訂閱訊息。如需如何從自訂元件發佈訊息至此元件的詳細資訊，請參閱[發佈/訂閱本機訊息](#)。

預設主題（本機發佈/訂閱）：sns/message

訊息接受下列屬性。輸入訊息必須是 JSON 格式。

request

要傳送至 Amazon SNS 主題的訊息相關資訊。

類型：object 包含下列資訊：

message

訊息的內容做為字串。

若要傳送 JSON 物件，請將它序列化為字串，並為 message_structure 屬性指定 json。

類型：string

subject

（選用）訊息的主旨。

類型：string

主旨可以是 ASCII 文字，最多 100 個字元。它必須以字母、數字或標點符號開頭。它不能包含換行符號或控制字元。

sns_topic_arn

(選用) 此元件發佈訊息之 Amazon SNS 主題的 ARN。指定此屬性以覆寫預設的 Amazon SNS 主題。

類型：string

message_structure

(選用) 訊息的結構。指定 json 傳送您在 content 屬性中序列化為字串的 JSON 訊息。

類型：string

有效值：json

id

請求的任意 ID。使用此屬性將輸入請求映射至輸出回應。當您指定此屬性時，元件會將回應物件中的 id 屬性設定為此值。

類型：string

 Note

訊息大小上限為 256 KB。

Example範例輸入：字串訊息

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example範例輸入：JSON 訊息

```
{
  "request": {
```

```
"subject": "Message subject",
"message": "{ \"default\": \"Message data\" }",
"message_structure": "json"
},
"id": "request123"
}
```

輸出資料

此元件預設會將回應發佈為下列 MQTT 主題的輸出資料。您必須將此主題指定為[舊版訂閱路由器元件](#)組態subject中的。如需如何在自訂元件中訂閱此主題訊息的詳細資訊，請參閱[發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

預設主題 (AWS IoT Core MQTT)：sns/message/status

Example範例輸出：成功

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example範例輸出：失敗

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

本機日誌檔案

此元件使用以下日誌檔案。

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
```

授權

此元件包含下列第三方軟體/授權：

- [適用於 Python \(Boto3\) 的 AWS SDK](#)/Apache License 2.0
- [botocore](#)/Apache License 2.0
- [dateutil](#)/PSF License
- [docutils](#)/BSD 授權、GNU 一般公有授權 (GPL)、Python 軟體基金會授權、公有網域
- [jmespath](#)/MIT License
- [s3transfer](#)/Apache License 2.0
- [urllib3](#)/MIT License

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.10	針對 Greengrass nucleus 2.15.0 版更新版本。
2.1.9	針對 Greengrass nucleus 2.14.0 版更新版本。
2.1.8	針對 Greengrass nucleus 2.13.0 版更新版本。
2.1.7	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.6	針對 Greengrass nucleus 2.11.0 版更新版本。

版本	變更
2.1.5	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.4	針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.3	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.2	針對 Greengrass nucleus 2.7.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.6.0 版更新版本。
2.1.0	新功能 • 新增對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱連線至連接埠 443 或透過網路代理及讓核心裝置信任 HTTPS 代理。
2.0.8	針對 Greengrass nucleus 2.5.0 版更新版本。
2.0.7	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.6	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.5	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.4	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.3	初始版本。

串流管理員

串流管理員元件 (`aws.greengrass.StreamManager`) 可讓您處理資料串流，以 AWS 雲端 從 Greengrass 核心裝置傳輸到。

如需如何在自訂元件中設定和使用串流管理員的詳細資訊，請參閱 [管理 Greengrass 核心裝置上的資料串流](#)。

主題

- [版本](#)
- [類型](#)

- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.2.x 版本
- 2.1.x
- 2.0.x

Note

如果您使用串流管理員將資料匯出至雲端，則無法將串流管理員元件的 2.0.7 版升級至 v2.0.8 和 v2.0.11 之間的版本。如果您是第一次部署串流管理員，強烈建議您部署最新版本的串流管理員元件。

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- [字元交換角色](#) 必須允許存取您搭配串流管理員使用的 AWS 雲端目的地。如需詳細資訊，請參閱：
 - [the section called “AWS IoT Analytics 頻道”](#)
 - [the section called “Amazon Kinesis 資料串流”](#)
 - [the section called “AWS IoT SiteWise 資產屬性”](#)
 - [the section called “Amazon S3 物件”](#)
- 支援在 VPC 中執行串流管理員元件。若要在 VPC 中部署此元件，需要下列項目。
 - 串流管理員元件必須與您發佈資料的 AWS 服務具有連線能力。
 - Amazon S3：com.amazonaws.*region*.s3
 - Amazon Kinesis Data Streams：com.amazonaws.*region*.kinesis-streams
 - AWS IoT SiteWise: com.amazonaws.*region*.iotsitewise.data
 - 如果您將資料發佈至 us-east-1 區域中的 Amazon S3，此元件預設會嘗試使用 S3 全域端點；不過，此端點無法透過 Amazon S3 VPC 介面端點使用。如需詳細資訊，請參閱 [AWS PrivateLink Amazon S3 的限制](#)。若要解決此問題，您可以從下列選項中選擇。
 - 透過在 `中設定`，將串流管理員元件設定為使用 us-east-1 區域中的區域 S3 - Daws.s3UseUsEast1RegionalEndpoint=regional 端點 JVM_ARGS。
 - 建立 Amazon S3 閘道 VPC 端點，而不是 Amazon S3 介面 VPC 端點。S3 閘道端點支援存取 S3 全域端點。如需詳細資訊，請參閱 [建立閘道端點](#)。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱 [允許透過代理或防火牆的裝置流量](#)。

Endpoint	連線埠	必要	說明
iotanalytics. <i>region</i> .amazonaws.com	443	否	如果您將資料發佈至 <code>region</code> ，則為必要項目 AWS IoT Analytics。

Endpoint	連線埠	必要	說明
kinesis. <i>region</i> .amazonaws.com	443	否	如果您將資料發佈至 Firehose，則為必要項目。
data.iots itewise. <i>region</i> .amazonaws.com	443	否	如果您將資料發佈至，則為必要項目 AWS IoT SiteWise。
*.s3.amazonaws.com	443	否	如果您將資料發佈至 S3 儲存貯體，則為必要項目。 您可以將取代*為您發佈資料的每個儲存貯體的名稱。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

2.1.3 – 2.2.1

下表列出此元件 2.1.3 至 2.2.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <3.0.0	軟式
字符交換服務	>=0.0.0	硬式

2.1.11 – 2.1.12

下表列出此元件 2.1.11 至 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.13.0	軟式
字符交換服務	>=0.0.0	硬式

2.1.9 – 2.1.10

下表列出此元件 2.1.9 至 2.1.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.12.0	軟式
字符交換服務	>=0.0.0	硬式

2.1.5 – 2.1.8

下表列出此元件 2.1.5 至 2.1.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.11.0	軟式
字符交換服務	>=0.0.0	硬式

2.1.2 – 2.1.4

下表列出此元件 2.1.2 至 2.1.4 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.10.0	軟式
權杖交換服務	>=0.0.0	硬式

2.1.1

下表列出此元件 2.1.1 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.9.0	軟式
權杖交換服務	>=0.0.0	硬式

2.1.0

下表列出此元件 2.1.0 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.8.0	軟式
權杖交換服務	>=0.0.0	硬式

2.0.15

下表列出此元件 2.0.15 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.7.0	軟式

相依性	相容版本	相依性類型
權杖交換服務	>=0.0.0	硬式

2.0.13 and 2.0.14

下表列出此元件 2.0.13 和 2.0.14 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.6.0	軟式
權杖交換服務	>=0.0.0	硬式

2.0.11 and 2.0.12

下表列出此元件 2.0.11 和 2.0.12 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.5.0	軟式
權杖交換服務	>=0.0.0	硬式

2.0.10

下表列出此元件 2.0.10 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.4.0	軟式
字符交換服務	>=0.0.0	硬式

2.0.9

下表列出此元件 2.0.9 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.3.0	軟式
字符交換服務	>=0.0.0	硬式

2.0.8

下表列出此元件 2.0.8 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.0 <2.2.0	軟式
權杖交換服務	>=0.0.0	硬式

2.0.7

下表列出此元件 2.0.7 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.0.3 <2.1.0	軟式
字符交換服務	>=0.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

STREAM_MANAGER_STORE_ROOT_DIR

(選用) 用於存放串流之本機目錄的絕對路徑。此值必須以正斜線開頭 (例如 /data)。

您必須指定現有的資料夾，而且[執行串流管理員元件的系統使用者](#)必須具有讀取和寫入此資料夾的許可。例如，您可以執行下列命令來建立和設定資料夾 `/var/greengrass/streams`，您可以將其指定為串流管理員根資料夾。這些命令允許預設系統使用者 `ggc_user` 讀取和寫入此資料夾。

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

預設：`/greengrass/v2/work/aws.greengrass.StreamManager`

`STREAM_MANAGER_SERVER_PORT`

(選用) 用於與串流管理員通訊的本機連接埠號碼。

您可以指定 0 使用隨機的可用連接埠。

預設：8088

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(選用) 用戶端必須先進行身分驗證，才能與串流管理員互動。串流管理員 SDK 控制用戶端和串流管理員之間的互動。此參數會決定哪些用戶端可以呼叫 Stream Manager SDK 來使用串流。如需詳細資訊，請參閱[串流管理員用戶端身分驗證](#)。

如果您指定 `true`，串流管理員 SDK 僅允許 Greengrass 元件做為用戶端。

如果您指定 `false`，串流管理員 SDK 可讓核心裝置上的所有程序成為用戶端。

預設：`true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(選用) 串流管理員可用來匯出資料的平均最大頻寬 (以每秒 KB 為單位)。

預設：無限制

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(選用) 串流管理員可用來匯出資料的作用中執行緒數目上限。

最佳大小取決於您的硬體、串流磁碟區和規劃的匯出串流數量。如果匯出速度很慢，您可以調整此設定，找出適合您硬體和商務案例的最佳大小。核心裝置硬體的 CPU 和記憶體是限制因素。首先，您可以嘗試將此值設定為等同於裝置上處理器核心的數量。

請小心不要設定高於硬體可支援的大小。每個串流都會耗用硬體資源，因此請嘗試限制受限裝置上的匯出串流數量。

預設：5 個執行緒

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

(選用) 分段中組件的大小下限 (以位元組為單位) 上傳至 Amazon S3。串流管理員使用此設定和輸入檔案的大小，來判斷如何在分段 PUT 請求中批次處理資料。

 Note

串流管理員使用串流 `sizeThresholdForMultipartUploadBytes` 屬性來判斷要匯出至 Amazon S3 做為單一或分段上傳。AWS IoT Greengrass 元件可以在建立匯出至 Amazon S3 的串流時設定此閾值。

預設：5242880 (5 MB)。這也是最小值。

`LOG_LEVEL`

(選用) 元件的記錄層級。從下列日誌層級中選擇，此處依層級順序列出：

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

`JVM_ARGS`

(選用) 啟動時要傳遞給串流管理員的自訂 Java 虛擬機器引數。依空格分隔多個引數。

僅限必須覆寫 JVM 使用的預設設定時，才能使用此參數。例如，如果您計劃匯出大量串流，可能需要增加預設堆積大小。

`startupTimeoutSeconds`

(選用) 元件啟動的時間上限，以秒為單位。ERRORED 如果元件超過此逾時，元件的狀態會變更為。

預設：120

Example範例：組態合併更新

下列範例組態指定 使用非預設連接埠。

```
{  
  "STREAM_MANAGER_SERVER_PORT": "18088"  
}
```

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.2.1	錯誤修正和改善 - 修正串流管理員無法將訊息匯出至 Kinesis Data Streams 目的地的問題。 - 串流管理員匯出至 Kinesis Data Streams 目的地效能的其他改進。
2.2.0	新功能 - 新增啟動逾時的新組態金鑰。預設值為 120 秒。 - 新增 Greengrass nucleus lite 的配方支援。
2.1.13	錯誤修正和改善 支援的 FIPS 端點 AWS IoT SiteWise
2.1.12	錯誤修正和改善 更新使用登入資料的順序，以便 Greengrass 登入資料優先於 AWS 服務請求。
2.1.11	針對 Greengrass nucleus 2.12.0 版更新版本。
2.1.10	錯誤修正和改善 修正 HTTPS 代理組態不信任 Greengrass 憑證授權機構 (CA) 憑證鏈的問題。
2.1.9	針對 Greengrass nucleus 2.11.0 版更新版本。
2.1.8	錯誤修正和改善 修正串流管理員無限重試 SiteWise 匯出因而失敗的問題 <code>InvalidRequestException</code>。
2.1.7	錯誤修正和改善 修正串流管理員無法正確讀取代理組態的問題。

版本	變更
2.1.6	錯誤修正和改善 修正在特定 ARMv8 處理器上啟動時可能導致當機的問題，包括 Jetson Nano。
2.1.5	針對 Greengrass nucleus 2.10.0 版更新版本。
2.1.4	錯誤修正和改善 - 修正 <code>ConflictingOperationException</code> SiteWise API 在單一批次傳回中具有相同時間戳記之相同屬性資產的項目導致串流管理員持續重試的問題。 - 將預設連線逾時從 3 秒更新為 1 分鐘。
2.1.3	錯誤修正和改善 修正以 SYSTEM 使用者身分執行時 Windows 作業系統上的啟動問題。
2.1.2	錯誤修正和改善 - 修正使用非英文語言的 Windows 作業系統問題。 - 針對 Greengrass nucleus 2.9.0 版更新版本。
2.1.1	針對 Greengrass nucleus 2.8.0 版更新版本。
2.1.0	新功能 - 更新此元件以自動將遙測指標傳送至 Amazon EventBridge。如需詳細資訊，請參閱從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料。 此功能需要 2.7.0 版或更新版本的 Greengrass 核元件。 - 針對 Greengrass nucleus 2.7.0 版更新版本。
2.0.15	針對 Greengrass nucleus 2.6.0 版更新版本。
2.0.14	此版本包含錯誤修正和改善。
2.0.13	針對 Greengrass nucleus 2.5.0 版更新版本。

版本	變更
2.0.12	錯誤修正和改善 修正無法將串流管理員 v2.0.7 升級至 v2.0.8 和 v2.0.11 之間的版本的問題。如果您使用串流管理員將資料匯出至雲端，您現在可以升級至 v2.0.12。
2.0.11	針對 Greengrass nucleus 2.4.0 版更新版本。
2.0.10	針對 Greengrass nucleus 2.3.0 版更新版本。
2.0.9	針對 Greengrass nucleus 2.2.0 版更新版本。
2.0.8	針對 Greengrass nucleus 2.1.0 版更新版本。
2.0.7	初始版本。

系統日誌轉寄站

系統日誌轉送器 (`aws.greengrass.SystemLogForwarder`) 會使用 Amazon CloudWatch CloudWatch。

Important

此元件只會轉送執行時間期間產生的 `systemd-journald` 日誌。如需 `systemd-journald` 日誌的詳細資訊，請參閱 [systemd-journald](#) 和 [journalctl](#)。

Note

此元件需要特定許可，才能建立和管理 CloudWatch 日誌群組和串流。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)

- [要求](#)
- [端點和連接埠](#)
- [相依性](#)
- [Configuration](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.1.x
- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件必須安裝在以系統為基礎的 Linux 系統上。

要求

此元件有下列需求：

元件需要存取才能在 CloudWatch 中建立日誌和串流群組，以及執行 PutLogEvents HTTP 呼叫的許可。您必須至少將下列政策許可新增至 Greengrass 裝置的角色別名：

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

如需詳細資訊，請參閱 System Log Forwarder [Github](#) 頁面。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱 [允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
logs. <i>region</i> .amazonaws.com	443	否	如果您將日誌寫入 CloudWatch Logs，則為必要項目。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件 [發行版本的](#) 相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以 [在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

2.1.x

下表列出此元件 2.1.x 版的相依性。

相依性	相容版本	相依性類型
權杖交換服務	$\geq 2.0.0$	硬式
Greengrass nucleus lite	$\geq 2.3.0$	軟式

2.0.x

下表列出此元件 2.0.x 版的相依性。

相依性	相容版本	相依性類型
權杖交換服務	$\geq 2.0.0$	硬式

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

2.0.x-2.1.x

`maxUploadIntervalSec`

系統日誌轉送器將嘗試上傳日誌的最長期間。由於日誌轉送器會在記憶體填滿時上傳日誌，因此其上傳頻率可能仍高於設定的最大節奏。

`maxRetriesCount`

系統日誌轉寄站嘗試重試暫時性 HTTP 錯誤的次數。

`bufferCapacity`

記憶體內日誌儲存的環形緩衝區大小。

`logGroup`

CloudWatch 中的日誌路徑。

logStream

CloudWatch logStream。

filters

核心裝置的篩選條件組態映射。

services

System Log Forwarder 將用於判斷上傳哪些日誌的服務名稱篩選條件清單。只有在源自 的服務符合此清單中的至少一個篩選條件時，才會上傳日誌。此清單中的篩選條件可能是服務名稱必須完全相符的字串，或是結尾為 * 的字串，這表示字首必須相符。

預設：[ggl.*]

Important

只有在源自 的服務符合此清單中的至少一個篩選條件時，才會上傳日誌。

Note

使用值 * 將包含所有可用的服務。

Example範例組態：

以下範例會依 Greengrass Nucleus Lite 中包含的所有服務篩選日誌。

```
{
  "maxUploadIntervalSec": 300,
  "maxRetriesCount": 3,
  "bufferCapacity": 1048576,
  "logGroup": "greengrass/systemLogs",
  "logStream": "deviceName",
  "filters": {
    "services": ["ggl.*"]
  }
}
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.1.0	錯誤修正和改善 - 更新元件配方以正確支援 Greengrass 核。 - 改善在沒有要上傳的日誌時的記錄輸出。 - 一般錯誤修正與改進。
2.0.1	錯誤修正和改善 更新元件配方以正確支援 aarch64 (arm64) 系統。
2.0.0	初始版本。

Systems Manager 代理程式

Agent AWS Systems Manager 元件 (`aws.greengrass.SystemsManagerAgent`) 會安裝 Systems Manager Agent，因此您可以使用 Systems Manager 管理核心裝置。Systems Manager 是一項 AWS 服務，可讓您用來檢視和控制上的基礎設施 AWS，包括 Amazon EC2 執行個體、內部部署伺服器和虛擬機器 VMs)，以及邊緣裝置。Systems Manager 可讓您檢視操作資料、自動化操作任務，以及維護安全性和合規性。如需詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的[什麼是 AWS Systems Manager ?](#) 以及[關於 Systems Manager 代理程式](#)。

Systems Manager 工具和功能稱為功能。Greengrass 核心裝置支援所有 Systems Manager 功能。如需這些功能以及如何使用 Systems Manager 管理核心裝置的詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的 [Systems Manager 功能](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)

- [Configuration](#)
- [本機日誌檔案](#)
- [另請參閱](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件只能在 Linux 核心裝置上安裝。

要求

此元件有下列需求：

- 在 64 位元 Linux 平台上執行的 Greengrass 核心裝置：Armv8 (AArch64) 或 x86_64。
- 您必須擁有 Systems Manager 可以擔任的 AWS Identity and Access Management (IAM) 服務角色。此角色必須包含 [AmazonSSMManagedInstanceCore](#) 受管政策或定義同等許可的自訂政策。如需詳細資訊，請參閱 [《使用者指南》中的為邊緣裝置建立 IAM 服務角色](#)。AWS Systems Manager 部署此元件時，您必須指定此角色的SSMRegistrationRole組態參數名稱。
- [Greengrass 裝置角色](#)必須允許 ssm:AddTagsToResource和 ssm:RegisterManagedInstance動作。裝置角色也必須允許符合先前需求的 IAM 服務角色的 iam:PassRole動作。下列範例 IAM 政策會授予這些許可。

```
{
  "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": "*"
    }
  ]
}
```

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	連線埠	必要	說明
ec2messages. <i>region</i> .amazonaws.com	443	是	在 中與 Systems Manager 服務通訊 AWS 雲端。
ssm. <i>region</i> .amazonaws.com	443	是	將核心裝置註冊為

Endpoint	連線埠	必要	說明
			Systems Manager 受管節點。
ssmmessages. <i>region</i> .amazonaws.com	443	是	在 中與 Systems Manager 的功能 Session Manager 通訊 AWS 雲端。

如需詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的[參考：](#)
[ec2messages](#)、[ssmmessages](#) 和其他 API 呼叫。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

下表列出此元件 1.0.0 到 1.3.0 版的相依性。

相依性	相容版本	相依性類型
權杖交換服務	>=2.0.0 <3.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

SSMRegistrationRole

Systems Manager 可擔任的 IAM 服務角色，其中包含 [AmazonSSMManagedInstanceCore](#) 受管政策或定義同等許可的自訂政策。如需詳細資訊，請參閱 [《使用者指南》中的為邊緣裝置建立 IAM 服務角色](#)。AWS Systems Manager

SSMOverrideExistingRegistration

(選用) 如果核心裝置已執行以混合啟用註冊的 Systems Manager Agent，您可以覆寫裝置現有的 Systems Manager Agent 註冊。將此選項設定為 `true`，以使用此元件提供的 Systems Manager Agent 將核心裝置註冊為受管節點。

Note

此選項僅適用於使用混合啟用註冊的裝置。如果核心裝置在已安裝 Systems Manager Agent 且已設定執行個體描述檔角色的 Amazon EC2 執行個體上執行，Amazon EC2 執行個體的現有受管節點 ID 會以開頭 `i-`。安裝 Systems Manager 代理程式元件時，Systems Manager 代理程式會註冊新的受管節點，其 ID 開頭為 `mi-`，而不是 `i-`。然後，您可以使用 ID 開頭為 `mi-` 的受管節點，透過 Systems Manager 管理核心裝置。

預設：`false`

SSMResourceTags

(選用) 要新增至 Systems Manager 受管節點的標籤，此元件會為核心裝置建立。您可以使用這些標籤，透過 Systems Manager 管理核心裝置的群組。例如，您可以在具有您指定之標籤的所有裝置上執行命令。

指定清單，其中每個標籤都是具有 Key 和 的物件 Value。例如， 的下列值會 `SSMResourceTags` 指示此元件在核心裝置的受管節點 **richard-roe** 上將 **Owner** 標籤設定為 。

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

如果受管節點已存在且 `SSMOverrideExistingRegistration` 為 `false`，此元件會忽略這些標籤。

Example範例：組態合併更新

下列範例組態指定 使用名為 的服務角色SSMServiceRole，以允許核心裝置註冊並與 Systems Manager 通訊。

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

本機日誌檔案

Systems Manager Agent 軟體會將日誌寫入 Greengrass 根資料夾以外的資料夾。如需詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的[檢視 Systems Manager Agent 日誌](#)。

Systems Manager Agent 元件使用 shell 指令碼來安裝、啟動和停止 Systems Manager Agent。您可以在下列日誌檔案中找到這些指令碼的輸出。

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。*/greengrass/v2* 將取代為 AWS IoT Greengrass 根資料夾的路徑。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

另請參閱

- [使用 管理 Greengrass 核心裝置 AWS Systems Manager](#)

- 《AWS Systems Manager 使用者指南》中的[什麼是 AWS Systems Manager ?](#)
- 《AWS Systems Manager 使用者指南》中的[關於 Systems Manager Agent](#)

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.3.0	錯誤修正和改善 更新此元件以取得 代理程式版本 3.3.1142.0。
1.2.4	錯誤修正和改善 更新此元件以取得 代理程式版本 3.2.2303.0。
1.2.3	錯誤修正和改善 • 在 Greengrass 上以 Snap 新增代理程式元件安裝的重試次數。 • 更新客服人員元件的組態，以僅使用 Greengrass 中的 Onprem Identity。 • 只有在已安裝的 Agent 版本與 Greengrass SSM Agent 元件的版本不相符時，才會更新此元件以更新 Agent。
1.1.0	此版本包含錯誤修正和改善。
1.0.0	初始版本。

字符交換服務

字符交換服務元件 (aws.greengrass.TokenExchangeService) 提供 AWS 登入資料，可用來與自訂元件中的 AWS 服務互動。

字符交換服務會以本機伺服器的形式執行。此本機伺服器會使用您在 [Greengrass 核心核元件](#) 中設定 AWS IoT 的角色別名，連線至 AWS IoT 登入資料提供者。

元件提供兩個環境變數，AWS_CONTAINER_CREDENTIALS_FULL_URI 以及 AWS_CONTAINER_AUTHORIZATION_TOKEN。會 AWS_CONTAINER_CREDENTIALS_FULL_URI 定義此本機伺服器的 URI。當元件建立 AWS SDK 用戶端時，用戶端會辨識此 URI 環境變數，並使用中

的字符 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 來連線至字符交換服務並擷取 AWS 憑證。這可讓 Greengrass 核心裝置呼叫 AWS 服務操作。如需如何在自訂元件中使用此元件的詳細資訊，請參閱 [與 AWS 服務互動](#)。

Important

以這種方式取得 AWS 登入資料的支援已於 2016 年 7 月 13 日新增至 AWS SDKs。您的元件必須使用在該日期或之後建立的 AWS SDK 版本。如需詳細資訊，請參閱《Amazon Elastic Container Service 開發人員指南》中的 [使用支援的 AWS SDK](#)。

Note

使用 AWS 預設登入資料鏈結的元件如果存在，可能會探索並使用其他登入資料來源。在 Amazon Elastic Container Service (Amazon ECS) 容器憑證中 `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI`，將優先於 `AWS_CONTAINER_CREDENTIALS_FULL_URI`，這表示字符交換服務憑證可能無法在 Amazon Elastic Container Service (Amazon ECS) 中運作。如需詳細資訊，請參閱 [標準化憑證提供者](#)

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)

版本

此元件具有下列版本：

- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#) 會執行元件的生命週期指令碼。

如需詳細資訊，請參閱 [元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

相依性

此元件沒有任何相依性。

Configuration

此元件提供下列組態參數，您可以在部署元件時自訂這些參數。

port

用於字符交換服務連線的連接埠。字符交換服務會在連接埠組態變更後重新啟動。

credentialRetryInSec

當 Token Exchange Service 遇到登入資料請求錯誤時，以秒為單位指定重試間隔。

clientError

用戶端錯誤 (4xx HTTP 狀態碼) 的重試間隔，以秒為單位。

預設：120

有效值：10 至 42900

serverError

伺服器錯誤的重試間隔，以秒為單位 (5xx HTTP 狀態碼)。

預設：60

有效值：10 至 42900

unknownError

未知錯誤（連線錯誤和 4xx 和 5xx 範圍以外的 HTTP 狀態碼）的重試間隔，以秒為單位。

預設：300

有效值：10 至 42900

Example範例：組態合併更新

```
{
  "port": 2020,
  "credentialRetryInSec": {
    "clientError": 30,
    "serverError": 45,
    "unknownError": 60
  }
}
```

本機日誌檔案

此元件使用與 [Greengrass 核](#) 元件相同的日誌檔案。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
2.0.3	初始版本。

IoT SiteWise OPC UA 收集器

IoT SiteWise OPC UA 收集器元件 (aws.iot.SiteWiseEdgeCollectorOpcua) 可讓 AWS IoT SiteWise 閘道從本機 OPC UA 伺服器收集資料。

使用此元件，AWS IoT SiteWise 閘道可以連接到多個 OPC UA 伺服器。如需 AWS IoT SiteWise 閘道的詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[AWS IoT SiteWise 在邊緣使用](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [輸出資料](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 3.1.x 版本
- 3.0.x
- 2.6.x
- 2.5.x
- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- Greengrass 核心裝置必須在下列其中一個平台上執行：
 - os : Ubuntu 20.04 或更新版本
 - 架構 : x86_64 (AMD64) 或 ARMv8 (Aarch64)
 - os : Red Hat Enterprise Linux (RHEL) 8

架構：x86_64 (AMD64) 或 ARMv8 (Aarch64)

- os：Amazon Linux 2

架構：x86_64 (AMD64) 或 ARMv8 (Aarch64)

- os：Debian 11

架構：x86_64 (AMD64) 或 ARMv8 (Aarch64)

- os：Windows Server 2019 或更新版本

架構：x86_64 (AMD64)

- Greengrass 核心裝置必須允許輸出網路連線到 OPC UA 伺服器。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

下表列出此元件所有版本的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.3.0 <3.0.0	硬式
串流管理員	>2.0.10<3.0.0	硬式
秘密管理員	>=2.0.8 <3.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件沒有任何組態參數。

您可以使用 AWS IoT SiteWise 主控台或 API 來設定 IoT SiteWise OPC UA 收集器元件。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[步驟 4：新增資料來源 - 選用](#)。

輸入資料

此元件僅接受下列格式的資料，其他所有項目都會遭到忽略和捨棄。下表將 OPC UA 資料類型映射至其 SiteWise 對等項目。

SiteWise 資料類型	OPC UA 資料類型	Description
STRING	String	長度上限為 1024 位元組的字串。
	Guid	
	XmlElement	
INTEGER	SByte	帶正負號的 32 位元整數，範圍為 -2,147,483,648 to 2,147,483,647 。
	Byte	
	Int16	
	UInt16	
	Int32	
	UInt32*	
DOUBLE	Int64*	範圍為 -10^{100} to 10^{100} 和 IEEE 754 雙精度的浮點數。
	UInt32*	
	Float	
	Double	
BOOLEAN	Boolean	true 或 false

* 對於 OPC UA 資料類型 UInt32 和 Int64，INTEGER 如果 SiteWise 能夠代表其值，則其 SiteWise 資料類型將是 `INTEGER`，否則將是 `DOUBLE`。

輸出資料

此元件會將BatchPutAssetPropertyValue訊息寫入 AWS IoT Greengrass 串流管理員。如需詳細資訊，請參閱《AWS IoT SiteWise API 參考》中的 [BatchPutAssetPropertyValue](#)。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

授權

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

版本	變更
3.1.0	新功能 - 為每個資料來源新增節點探索程序的可設定性。 - 為每個資料來源新增預設資料變更觸發條件的可設定性。 錯誤修正和改善 - 修正安全漏洞。
3.0.3	錯誤修正和改善 - 修正同步大型組態時造成檔案截斷和同步失敗的問題。 - 修正因 AWS IoT Core 訂閱延遲而導致啟動程序延遲的問題。 - 修正暫時性中斷 OPC UA 伺服器連線時觸發完整探索的問題。
3.0.2	錯誤修正和改善 修正 <code>OpcUaCollector.IncomingValuesCount</code> 指標，以準確在計算中包含快照點。
3.0.1	錯誤修正和改善 - 修正安全漏洞。 - 修正 OPC UA 時間戳記處理的問題。新增偏好的時間戳記選擇，並在偏好的時間戳記無法使用時改善備用邏輯。 - 修正未指定目的地時，Classic 串流、V2 閘道 OPC UA 收集器組態無法預設為 Stream Manager 的問題。
3.0.0	新功能 除了 Classic 串流、V2 閘道（先前稱為自我託管 SiteWise Edge 閘道）之外，新增對啟用 MQTT 的 V3 閘道的支援。V2 SiteWise 如需詳細資訊，請參閱啟用 MQTT 的 AWS IoT SiteWise Edge V3 閘道。 - 允許 OPC UA 資料發佈至 AWS IoT Greengrass MQTT 5 代理程式 (EMQX)。 錯誤修正和改善 - 修正安全漏洞。

版本	變更
2.6.0	新功能 如果啟用擷取，則新增對擷取 Null 和 NaN 值的支援 AWS IoT SiteWise。若要檢視或修改 中的 Null 和 NaN 組態 AWS IoT SiteWise，請參閱 DescribeStorageConfiguration 和 PutStorageConfiguration APIs。
2.5.1	錯誤修正和改善 - 修正在快照任務執行時遇到錯誤時，取消未來快照任務的錯誤。 - 修正以下錯誤：如果與資料來源的 OPC UA 伺服器的連線中斷，則在重新啟動 OPC UA 收集器之後，資料來源組態更新才會持續存在。
2.5.0	新功能 - 新增資料來源選項，將簡單陣列和 DateTime 值轉換為字串。 - 新增屬性群組選項，以在從 OPC UA 伺服器收集資料時選取來源或伺服器時間戳記。 錯誤修正和改善 - 重設預設輪詢組態以使用來源時間戳記。
2.4.2	錯誤修正和改善 - 修正 OPC UA 伺服器探索期間可能多次探索節點的問題。 - 修正快照功能，以確保每個快照資料點的時間戳記都是新的。
2.4.1	錯誤修正和改善 - 修正與代理支援相關的問題。 - 修正執行緒清除失敗並造成資料封鎖的問題。
2.4.0	新功能 - 新增事件日誌，讓您更輕鬆地識別和修復問題。 錯誤修正和改善 - 修正連線至使用 OPC UA 規格 1.05 版的 OPC UA 伺服器時，造成憑證錯誤的問題。

版本	變更
2.3.0	新功能 • 新增對 Linux 上 Greengrass 核 HTTP 代理 組態的支援。 錯誤修正和改善 • 修正安全問題 (CVE-2019-19135)。
2.2.0	新功能 • 新增在 Linux ARMv8 架構上安裝 Data Collection Pack 的支援。 • Linux ARMv8 的最低需求： • 記憶體：4 GB • CPU：ARM Cortex-A72 或同等規格 錯誤修正和改善 • 改善節點探索程序中指標的記錄。 • 改善處理不支援的資料類型。 • 改善資料串流錯誤的記錄。
2.1.3	新功能 • 新增對 Windows Server 2019 或更新版本的支援。 錯誤修正和改善 • 改善在不支援的裝置上部署此元件時的錯誤訊息。

版本	變更
2.1.1	新功能 • 新增設定下列訂閱屬性的支援： • DataChangeTrigger - 您可以定義啟動資料變更提醒的條件。 • QueueSize - OPC-UA 伺服器上特定指標的佇列深度，其中監控項目的通知會排入佇列。 • PublishingIntervalMilliseconds - 建立訂閱時指定的發佈週期間隔（以毫秒為單位）。 • SnapshotFrequencyMilliseconds - 您可以設定快照頻率逾時設定，以確保 AWS IoT SiteWise Edge 擷取穩定的資料串流。 • 此版本支援擷取BAD品質資料，並根據下列資料品質篩選資料： • UNCERTAIN 品質資料 • BAD 品質資料 錯誤修正和改善 • 改善客戶指標。 • 修正連線至啟用加密的伺服器時有時會造成問題的安全性編碼。 • 修正屬性群組無法更新的問題。
2.0.3	錯誤修正和改進。
2.0.2	與邊緣同步資產優先順序的錯誤修正和改進。
2.0.1	初始版本。

另請參閱

- AWS IoT SiteWise 《使用者指南》中的[什麼是 AWS IoT SiteWise ?](#)。

IoT SiteWise OPC UA 資料來源模擬器

IoT SiteWise OPC UA 資料來源模擬器元件

(aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator) 會啟動產生範例資料的本機 OPC UA 伺服器。使用此 OPC UA 伺服器模擬 AWS IoT SiteWise 閘道上 [IoT SiteWise OPC UA 收集器元件](#) 讀取

的資料來源。然後，您可以使用此範例資料來探索 AWS IoT SiteWise 功能。如需 AWS IoT SiteWise 閘道的詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[AWS IoT SiteWise 在邊緣使用](#)。

主題

- [版本](#)
- [類型](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 1.0.x

類型

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- Greengrass 核心裝置必須能夠在本機主機上使用連接埠 4840。此元件的本機 OPC UA 伺服器會在此連接埠執行。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

下表列出此元件所有版本的相依性。

相依性	相容版本	相依性類型
Greengrass 核	>=2.3.0 <3.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

變更記錄

下表說明每個版本元件的變更。

版本	變更
1.0.0	初始版本。 新增對 Windows Server 2016 或更新版本的支援。

另請參閱

- AWS IoT SiteWise 《使用者指南》中的[什麼是 AWS IoT SiteWise ?](#)。

IoT SiteWise 發佈者

IoT SiteWise 發佈者元件 (aws.iot.SiteWiseEdgePublisher) 可讓 AWS IoT SiteWise 閘道將資料從邊緣匯出至 AWS 雲端。

如需 AWS IoT SiteWise 閘道的詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[AWS IoT SiteWise 在邊緣使用](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)

- [要求](#)
- [相依性](#)
- [Configuration](#)
- [輸入資料](#)
- [本機日誌檔案](#)
- [故障診斷和偵錯](#)
- [授權](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 4.1.x
- 4.0.x
- 3.2.x
- 3.1.x 版本
- 3.0.x
- 2.4.x 版本
- 2.3.x 版本
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- Greengrass 核心裝置必須在下列其中一個平台上執行：
 - os : Ubuntu 18.04 或更新版本
架構 : x86_64 (AMD64) 或 ARMv8 (Aarch64)
 - os : Red Hat Enterprise Linux (RHEL) 8
架構 : x86_64 (AMD64) 或 ARMv8 (Aarch64)
 - os : Amazon Linux 2
架構 : x86_64 (AMD64) 或 ARMv8 (Aarch64)
 - os : Debian 11
架構 : x86_64 (AMD64) 或 ARMv8 (Aarch64)
 - os : Windows Server 2019 或更新版本
架構 : x86_64 (AMD64)
- Greengrass 核心裝置必須連線至網際網路。
- Greengrass 核心裝置必須獲得授權才能執行 `iotsitewise:BatchPutAssetPropertyValue` 動作。如需詳細資訊，請參閱[授權核心裝置以與服務互動 AWS](#)。

Example許可政策

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*"
    }
  ]
}
```

```
]
}
```

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
<code>data.iotsitewise.<i>region</i>.amazonaws.com</code>	443	是	將資料發佈至 AWS IoT SiteWise。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本的](#)相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台](#)中檢視每個版本元件的相依性。在元件詳細資訊頁面上，尋找相依性清單。

下表列出此元件 2.0.x 至 2.2.x 版的相依性。

相依性	相容版本	相依性類型
Greengrass 核	<code>>=2.3.0<3.0.0</code>	硬式
串流管理員	<code>>=2.0.10<3.0.0</code>	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件沒有任何組態參數。

您可以使用 AWS IoT SiteWise 主控台或 API 來設定 IoT SiteWise 發佈者元件。如需詳細資訊，請參閱 AWS IoT SiteWise 《使用者指南》中的[步驟 3：設定發佈者 - 選用](#)。

輸入資料

此元件會從 AWS IoT Greengrass 串流管理員讀取PutAssetPropertyValueEntry訊息。如需詳細資訊，請參閱 AWS IoT SiteWise API 參考中的 [PutAssetPropertyValueEntry](#)。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

故障診斷和偵錯

此元件包含新的事件日誌，協助客戶識別和修正問題。日誌檔案與本機日誌檔案分開，位於下列位置。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/  
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs  
\IotSiteWisePublisherEvents.log
```

此日誌包含詳細資訊和故障診斷指示。故障診斷資訊與診斷一起提供，並描述如何修正問題，有時還包含進一步資訊的連結。診斷資訊包括下列項目：

- 嚴重性等級
- 時間戳記
- 其他事件特定資訊

Example範例日誌

```
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

授權

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

版本	變更
4.1.4	錯誤修正和改善 - IoT SiteWise 發佈者現在會將 IoT SiteWise 發佈者的記憶體和 CPU 用量等系統運作狀態指標報告給 CloudWatch。 - 更新錯誤報告至 <code>greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/AggregatePropertyAliases_1686346224654.log</code>，AWS IoT Greengrass 以便在啟動工作流程發生部分故障時重新啟動 IoT SiteWise 發佈者。
4.1.3	錯誤修正和改善 - 修正資料檢查點程序中的記憶體流失。 - 修正安全漏洞。
4.1.2	錯誤修正和改善 - 修正與同步大型組態相關的問題。 - 訂閱中的延遲 AWS IoT Core 不再影響啟動程序。 - 初始化檢查點的延遲不會再影響啟動程序。
4.1.1	錯誤修正和改善 - 修正在重新啟動發佈者後，透過 AWS IoT SiteWise 緩衝的 Amazon S3 目的地傳送資料時，導致 Classic 串流、V2 閘道上 CPU 和記憶體使用量過高的問題。 - 透過建立位於 <code>greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/log4j2.xml</code> 的選用檔案，新增覆寫預設元件記錄行為的支援。

版本	變更
4.1.0	新功能 - 新增命令列工具來管理內建的 EMQX MQTT 代理程式使用者。此界面可啟用使用者帳戶管理，並設定 MQTT 主題發佈和訂閱許可的存取控制政策。 錯誤修正和改善 - 修正計算，決定有多少唯一別名可以放入單一緩衝擷取 Parquet 檔案。
4.0.3	錯誤修正和改善 - 更新每個 AWS IoT SiteWise 緩衝擷取檔案的唯一別名上限，以符合 10,000 個雲端限制。 - 修正錯誤回報無效服務狀態的記錄 AWS IoT Greengrass 問題。 - 新增驗證，以防止 Classic 串流、V2 閘道中跨 Amazon S3 目的地組態重複的串流名稱。 - 修正非標準 AWS 分割區中 AWS IoT SiteWise 緩衝擷取的相容性問題。 - 修正導致重複資料擷取的問題。
4.0.2	錯誤修正和改善 - 修正在批次等待時間過期後阻止資料上傳的問題。 - 新增驗證，以確認啟用 MQTT 的 V3 閘道組態中的每個目的地都是唯一的。
4.0.1	錯誤修正和改善 - 修正當您停用最大批次等待時間組態 AWS IoT SiteWise 時，防止資料上傳至的問題。 - 修正 <code>IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker</code> 指標不正確地報告訂閱 MQTT 代理程式之唯一主題計數的問題。

版本	變更
4.0.0	新功能 除了 Classic 串流、V2 閘道（先前稱為自我託管 SiteWise Edge 閘道）之外，還新增對啟用 MQTT 的 V3 閘道的支援。V2 SiteWise - 新增從 AWS IoT Greengrass MQTT 5 代理程式 (EMQX) 訂閱和接收資料的功能。如需詳細資訊，請參閱啟用 MQTT 的 AWS IoT SiteWise Edge V3 閘道。 - 使用路徑篩選條件新增資料目的地的組態選項。如需詳細資訊，請參閱了解 AWS IoT SiteWise Edge 目的地的路徑篩選條件。 錯誤修正和改善 - 修正 PublishSuccessCount 指標報告負值的問題。 - 修正發佈者未在 100 秒內啟動並進入 BROKEN 狀態的問題。 - 修正安全漏洞。
3.2.0	新功能 - 如果啟用擷取，則新增對擷取 Null 和 NaN 值的支援 AWS IoT SiteWise。若要檢視或修改中的 Null 和 NaN 組態 AWS IoT SiteWise，請參閱DescribeStorageConfiguration 和 PutStorageConfiguration APIs。 錯誤修正和改善 - 修正導致檢查點資料庫檔案損毀的問題。 - 修正產生重複指標的問題。
3.1.4	 Warning 3.1.4 版已於 2025 年 2 月 20 日終止。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 修正離線後可能導致longer-than-expected的問題。

版本	變更
3.1.3	錯誤修正和改善 • 修正建立位於 的事件日誌檔案，<code>/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/IoTSiteWisePublisherEvents.log</code> 但未記錄任何事件的問題。 • 新增下列 CloudWatch 指標，以監控與 MQTT 代理程式的連線： • <code>IoTSiteWisePublisher.IsConnectedToMqttBroker</code> • <code>IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker</code> • <code>IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived</code> • <code>IoTSiteWisePublisher.MqttMessageReceivedSuccessCount</code> • <code>IoTSiteWisePublisher.MqttReceivedSuccessBytes</code> 如需這些指標的詳細資訊，請參閱AWS IoT Greengrass Version 2 閘道指標。 • 修正即使上傳 parquet 檔案至 S3 失敗，仍會呼叫 BatchCreateJob API 的問題。
3.1.2	錯誤修正和改善 • 修正 3.1.1 版中引入的高 CPU 用量問題。

版本	變更
3.1.1	 Warning 3.1.1 版已於 2024 年 3 月 12 日終止。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 • 新增額外的記錄，在發生錯誤時識別受影響的資料別名。 • 在擷取資料的存留期新增對 AWS IoT SiteWise API 限制的本機強制執行。 • 修正發佈者在有多個 Amazon S3 目的地時混淆 StreamManager 串流檢查點的問題。 • 修正發佈者如何從 StreamManager 串流讀取效能瓶頸。
3.1.0	新功能 • 新增支援，將資料作為 parquet 檔案發佈至 Amazon S3。 • 新增 AWS IoT SiteWise 緩衝擷取的支援。
3.0.0	錯誤修正和改善 • 修正與代理支援相關的問題。 新功能 • 支援從 MQTT 代理程式擷取資料。
2.4.1	錯誤修正和改善 • 啟用元件以使用 Java Corretto 11 版本 11.0.20.8.1 和更新版本。元件版本 2.4.0 和 2.3.3 顯示與 Java Corretto 版本 11.0.20.8.1 搭配使用時的 "Could not find or load main class" 錯誤訊息。
2.4.0	新功能 • 新增事件日誌，讓您更輕鬆地識別和修復問題。 錯誤修正和改善 • 改善發佈者檢查點復原。

版本	變更
2.3.3	錯誤修正和改善 改善支援高輸送量的能力。
2.3.2	錯誤修正和改善 修正下載發佈者組態時的 HTTP 代理支援。
2.3.1	新功能 - 新增在 Linux ARMv8 架構上安裝資料收集套件的支援。 - Linux ARMv8 的最低需求： - 記憶體：4 GB - CPU：ARM Cortex-A72 或同等規格
2.2.3	錯誤修正和改善 移除不在可重試例外清單中的一般例外狀況的重試。
2.2.2	錯誤修正和改善 透過 HTTP AWS IoT SiteWise 代理伺服器重新導入對 的資料上傳支援。
2.2.1	 Note 此版本不支援 HTTP 代理組態。2.2.2 版和更新版本重新推出此功能的支援。 新功能 將支援新增至此元件，以在上傳資料至 時切換壓縮 AWS IoT SiteWise。

版本	變更
2.2.0	 Note 此版本不支援 HTTP 代理組態。2.2.2 版和更新版本重新推出此功能的支援。 新功能 - 更新此元件以壓縮資料，再將其傳送至 AWS IoT SiteWise 服務。 - 在大多數情況下，相較於此元件的先前版本，此變更會將頻寬使用量減少 75%。 - 在大多數情況下，此變更最多可將 CPU 用量增加 5%。在處理大量資料的閘道上，此變更最多可將 CPU 用量增加 15%。 - 此變更不會影響 AWS IoT SiteWise 服務費用或服務配額用量。 - 新增對 Windows Server 2019 或更新版本的支援。 錯誤修正和改善 - 修正當檢查點檔案損毀時，此元件無法啟動的問題。
2.1.4	錯誤修正和改善 修正與 Java 第 8 版的相容性。
2.1.3	 Warning 此版本不再提供，但美國東部（俄亥俄）、加拿大（中部）和 AWS GovCloud（美國東部）區域除外。此元件版本需要執行 Java 版本 11 或更高版本。此版本的改善功能可在此元件的較新版本中使用。 錯誤修正和改善 - 改善在不支援的裝置上部署此元件時的錯誤訊息。 - 資料上傳失敗時記錄錯誤的更新。
2.1.2	錯誤修正和改善 更新以在資料過期時叫用過期的資料匯出功能。

版本	變更
2.1.1	錯誤修正和改善。
2.1.0	新功能 • 新增支援，以先將最新的資料發佈至雲端。 • 新增不將過期資料發佈至雲端的支援。 • 新增在本機存放過期資料的支援。 錯誤修正和改善 • 減少磁碟 I/O 和對應的延遲。
2.0.2	錯誤修正和改善。
2.0.1	初始版本。

另請參閱

- AWS IoT SiteWise 《使用者指南》中的[什麼是 AWS IoT SiteWise ?](#)。

IoT SiteWise 處理器

IoT SiteWise 處理器元件 (`aws.iot.SiteWiseEdgeProcessor`) 可讓 AWS IoT SiteWise Classic 串流、V2 閘道在邊緣處理資料。

透過此元件，AWS IoT SiteWise 閘道可以使用資產模型和資產來處理閘道裝置上的資料。如需 AWS IoT SiteWise 閘道的詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[AWS IoT SiteWise 在邊緣使用](#)。

Note

資料處理套件 (DPP) 功能自 2025 年 11 月 7 日起將不再開放給新客戶使用。如果您想要使用 DPP，請在該日期之前註冊。現有客戶可以繼續正常使用該服務。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[資料處理套件可用性變更](#)。

主題

- [版本](#)
- [Type](#)
- [作業系統](#)
- [要求](#)
- [相依性](#)
- [Configuration](#)
- [本機日誌檔案](#)
- [授權](#)
- [變更記錄](#)
- [另請參閱](#)

版本

此元件具有下列版本：

- 3.5.x
- 3.4.x
- 3.3.x
- 3.2.x
- 3.1.x 版本
- 3.0.x
- 2.2.x 版本
- 2.1.x
- 2.0.x

Type

此元件是一般元件 (aws.greengrass.generic)。 [Greengrass 核](#)會執行元件的生命週期指令碼。

如需詳細資訊，請參閱[元件類型](#)。

作業系統

此元件可以安裝在執行下列作業系統的核心裝置上：

- Linux
- Windows

要求

此元件有下列需求：

- Greengrass 核心裝置必須在下列其中一個平台上執行：

- os : Ubuntu 20.04 或更新版本

架構 : x86_64 (AMD64)

- os : Red Hat Enterprise Linux (RHEL) 8

架構 : x86_64 (AMD64)

- os : Amazon Linux 2

架構 : x86_64 (AMD64)

- os : Windows Server 2019 或更新版本

架構 : x86_64 (AMD64)

- os : Debian 11 (Bullseye) 或更新版本

架構 : x86_64 (AMD64)

- Greengrass 核心裝置必須允許連接埠 443 上的傳入流量。
- Greengrass 核心裝置必須允許連接埠 443 和 8883 上的傳出流量。
- 下列連接埠保留供 使用 AWS IoT SiteWise：
80、443、3001、4569、4572、8000、8081、8082、8084、8085、8086、8445、9000、9500、11080
和 50010。使用保留連接埠處理流量可能會導致連線終止。

Note

只有此元件的 2.0.15 版和更新版本才需要連接埠 8087。

- [Greengrass 裝置角色](#)必須具有允許您在 AWS IoT Greengrass V2 裝置上使用 AWS IoT SiteWise 閘道的許可。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[需求](#)。

端點和連接埠

除了基本操作所需的端點和連接埠之外，此元件還必須能夠對下列端點和連接埠執行傳出請求。如需詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。

Endpoint	站點	必要	Description
<code>model.iotsitewise. <i>region</i>.amazonaws.com</code>	443	是	取得 AWS IoT SiteWise 資產和資產模型的相關資訊。
<code>edge.iotsitewise. <i>region</i>.amazonaws.com</code>	443	是	取得核心裝置的 AWS IoT SiteWise 閘道組態的相關資訊。
<code>ecr.<i>region</i>.amazonaws.com</code>	443	是	從 Amazon Elastic Container Registry 下載 AWS IoT SiteWise Edge 閘道 Docker 映像。
<code>iot.<i>region</i>.amazonaws.com</code>	443	是	為您的取得裝置端點 AWS 帳戶。

Endpoint	站點	必要	Description
sts. <i>region</i> .amazonaws.com	443	是	取得的 ID AWS 帳戶。
monitor.iotsitewis e. <i>region</i> .amazonaws.com	443	否	如果您在 核心裝置 上存取 AWS IoT SiteWise Monitor 入 口網站， 則為必要項 目。

相依性

當您部署元件時，AWS IoT Greengrass 也會部署其相依性的相容版本。這表示您必須符合元件及其所有相依性的要求，才能成功部署元件。本節列出此元件[發行版本](#)的相依性，以及為每個相依性定義元件版本的語意版本限制。您也可以[在 AWS IoT Greengrass 主控台中檢視每個版本元件的相依性](#)。在元件詳細資訊頁面上，尋找相依性清單。

下表列出此元件 2.0.x 至 2.1.x 版的相依性。

相依性	相容版本	相依性類型
權杖交換服務	>=2.0.3 <3.0.0	硬式
串流管理員	>=2.0.10 <3.0.0	硬式
Greengrass CLI	>=2.3.0 <3.0.0	硬式

如需元件相依性的詳細資訊，請參閱[元件配方參考](#)。

Configuration

此元件沒有任何組態參數。

本機日誌檔案

此元件使用以下日誌檔案。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

檢視此元件的日誌

- 在核心裝置上執行下列命令，以即時檢視此元件的日誌檔案。將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 AWS IoT Greengrass 根資料夾的路徑。

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
```

授權

此元件包含下列第三方軟體/授權：

- Apache-2.0
- MIT
- BSD-2-Clause
- BSD-3-Clause
- CDDL-1.0
- CDDL-1.1
- ISC

- Zlib
- GPL-3.0-with-GCC-exception
- 公有網域
- Python-2.0
- Unicode-DFS-2015
- BSD-1-Clause
- OpenSSL
- EPL-1.0
- EPL-2.0
- GPL-2.0-with-classpath-exception
- MPL-2.0
- CC0-1.0
- JSON

此元件根據 [Greengrass 核心軟體授權合約](#) 發行。

變更記錄

下表說明每個版本元件的變更。

版本	變更
3.5.1	新功能 如果啟用擷取，則新增對擷取 Null 和 NaN 值的支援 AWS IoT SiteWise。若要檢視或修改 中的 Null 和 NaN 組態 AWS IoT SiteWise，請參閱 DescribeStorageConfiguration 和 PutStorageConfiguration APIs。 錯誤修正和改善 更新相依性以解決潛在的安全漏洞。
3.4.0	新功能 • 新增選用的 HTTP 和 HTTPS 代理支援，以透過需要自訂憑證的代理伺服器啟用閘道通訊。如需設定 AWS IoT Greengrass 核心 HTTP 代理的詳細資訊，請參閱 連線至連接埠 443 或透過網路代理。如需 SiteWise

版本	變更
	Edge 特定代理支援的詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的管理 AWS IoT SiteWise Edge 代理支援的信任存放區。 新增可設定的工作階段逾時設定，以管理 AWS OpsHub 和 SiteWise Edge APIs 的閒置期間。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的設定 AWS IoT SiteWise Edge 的工作階段逾時。 效能改進 將傳入資料到達邊緣裝置儲存的時間從 5 秒縮短為不到 1 秒。資料上傳至的延遲 AWS IoT SiteWise 保持不變。
3.3.1	新功能 新增對 SiteWise Edge APIs選用 CORS 支援，增強跨來源資源共用功能。此功能可改善 Web 應用程式與 APIs 互動的彈性。
3.3.0	效能改進 - 最佳化快取重新整理機制，只需重新整理新資產或更新資產的項目，即可減少資產同步之間的 AWS IoT SiteWise I/O 用量。 - 減少記憶體使用量，以使用大量同步資產屬性來維護快取。 錯誤修正和改善 - 抑制在沒有擷取錯誤時擷取個別屬性值的日誌，這會降低高擷取率期間的日誌雜訊。 - 針對特定日誌項目使用人類可讀格式，改善日誌可讀性。 - 新增對 Java 17 和更新版本的支援。

版本	變更
3.2.1	錯誤修正和改善 • 修正 AWS IoT SiteWise API 呼叫未與 SiteWise Edge 同步分頁的問題。 • 修正不再發佈 <code>MessageRemaining.SiteWise_Edge_Stream</code> 指標的問題。 • 新增下列 CloudWatch 指標，以監控與 MQTT 代理程式的連線。 • <code>IoTSiteWiseProcessor.IsConnectedToMqttBroker</code> • <code>IoTSiteWiseProcessor.NumberOfSubscriptionsToMqttBroker</code> • <code>IoTSiteWiseProcessor.NumberOfUniqueMqttTopicsReceived</code> • <code>IoTSiteWiseProcessor.MqttMessageReceivedSuccessCount</code> • <code>IoTSiteWiseProcessor.MqttReceivedSuccessBytes</code> 如需這些指標的詳細資訊，請參閱 AWS IoT Greengrass Version 2 閘道指標。

版本	變更
3.2.0	效能改進 • 最佳化 API 服務以擁有較小的記憶體使用量，並需要較少的磁碟空間來安裝 • 這可為整個元件提供 2 GB 的初始記憶體用量減少（現在在啟動時使用 7.5 GB 的記憶體，但仍建議 16 GB），以及 500 MB 的下載大小減少（現在需要 1.4 GB 的下載）。 新功能 • <code>GetAssetPropertyValueAggregates</code> API 現在支援邊緣的 15 分鐘彙總時段。 • 連接埠 8081 和 8082 不再需要可供此元件正確執行。  Note AWS IoT SiteWise 資料平面 APIs 的本機端點，例如 <code>get-asset-property-value</code>，正從變更為 <code>http://localhost:8081</code> <code>http://localhost:11080/data</code>。AWS IoT SiteWise 控制平面 APIs 的本機端點，例如 <code>list-asset-models</code>，正從變更為 <code>http://localhost:11080</code> <code>http://localhost:11080/control</code>。AWS 一律建議您使用 SiteWise Edge 閘道 HTTPS 端點。這些端點尚未變更。 錯誤修正和改善 • 如果先前的同步中斷，從同步現在 AWS IoT SiteWise 會將資源轉換為有效狀態。這將修正某些資源在強制重新啟動後損毀的問題。 • 修正在同步期間修改資源時，可能在邊緣損毀的罕見條件。如果偵測到此條件，同步現在將會失敗，並且將在下一次同步中重試資源。 • 修正可能允許在外部呼叫 APIs 的 HTTP 端點的問題。現在只能使用 HTTPS 來呼叫本機迴路地址以外的 APIs。 • <code>ListAssets</code> API 現在會顯示存放在邊緣之資產的資產階層。 • 修正資料處理套件無法在 Windows 上重新啟動、升級或降級的問題。

版本	變更
	修正 Windows 作業系統的 Data Processing Pack 中的錯誤，讓客戶無法使用登入資料與 MQTT 中介裝置連線。
3.1.3	錯誤修正和改善 - 修正某些資源實際失敗時，資料處理套件錯誤回報成功同步的問題。 - 允許多個資產具有相同的名稱，只要它們沒有相同的父項。
3.1.1	錯誤修正和改善 - 修正 SigV4 請求因時區不相符而失敗的問題。 - 修正轉換和指標屬性在重新啟動後依賴屬性時停止計算的問題。 - 啟用自訂 Stream Manager 連接埠組態的支援。 - 修正同步至邊緣的屬性可能會停止更新的問題。
3.1.0	錯誤修正和改善 修正 ListAssetModels API 無法產生下一個字符的問題。
3.0.0	新功能 支援從 MQTT 代理程式擷取資料。
2.2.1	錯誤修正和改善 - 調整同步程序，讓控制平面資料儲存與雲端運作方式更一致。這稍微影響升級。  Note 在 2.2.1 版或更新版本上同步的控制平面資料與舊版不相容。若要降級至先前的版本，您需要完成新的安裝。這不會影響升級，在舊版上同步的資料將適用於 2.2.1 版。 - 對 AWS 登入資料鏈進行其他修改，以排定 AWS IoT Greengrass V2 登入資料的優先順序。

版本	變更
2.1.37	錯誤修正和改善 - 棄用 <code>dependency-routing-service</code> 程序，並將其功能移至 <code>property-state-service</code> 程序，以減少來自通訊程序的資源使用量。 - 將 <code>get-asset-property-value-history</code> API 的最大結果限制提高到 20,000，以符合所使用的限制 AWS IoT SiteWise。 - 修正未指定結果上限時，未在 <code>get-asset-property-value-history</code> API 的分頁結果中提供下一個字符的問題。
2.1.35	錯誤修正和改善 - 修改 AWS 登入資料鏈以排定登入資料的優先順序 AWS IoT Greengrass。 - 修正在部署做為 AWS IoT 物件群組的一部分時，帳戶偵測的問題。
2.1.34	錯誤修正和改善 - 調整指標/轉換運算，以在 Linux 上使用多執行緒。Windows 會繼續執行單一執行緒運算，以確保相容性。 - 修正某些運算時段缺少指標運算的問題。
2.1.33	錯誤修正和改善 修正向 Greengrass 主控台回報錯誤狀態的問題。
2.1.32	錯誤修正和改善 新增對自訂使用者名稱和群組的支援。
2.1.31	錯誤修正和改善 新增支援，以計算在 中建模的資料的時間加權平均值和時間加權標準差 AWS IoT SiteWise。
2.1.29	錯誤修正和改善 新增支援以篩選邊緣功能上的資產。
2.1.28	錯誤修正和改善 最佳化資源同步，讓大量資產能夠從 同步 AWS 雲端 到邊緣。

版本	變更
2.1.24	錯誤修正和改善 修正第二次同步資源時導致儀表板消失的問題。
2.1.23	錯誤修正和改善 - 新增 <code>aws.iot.SiteWiseEdgeProcessor</code> 安裝程序的逾時，以避免在網際網路連線速度緩慢時安裝失敗。 - 最佳化資源同步，以提高雲端和邊緣之間的同步效率。
2.1.21	 Warning 從 2.0.x 升級到 2.1.x 會導致本機資料遺失。 新功能 - 新增對 Windows Server 2019 或更新版本的支援。 - 移除 Linux 作業系統的 docker。
2.0.16	此版本包含錯誤修正和改善。
2.0.15	錯誤修正和改善 - 將此元件用於資源同步 API 操作的連接埠從 8085 變更為 8087。因此，此元件現在需要連接埠 8087 才能使用。此元件仍然需要連接埠 8085 才能使用。 - 更新 AWS OpsHub 身分驗證，以在登入期間拒絕未經授權的使用者，而不是當使用者嘗試呼叫 API 操作時。
2.0.14	此版本包含錯誤修正和改善。
2.0.13	錯誤修正和改善 修正問題，讓此元件向 Amazon CloudWatch 指標報告資料時，現在可正確指出哪些資料未建模。

版本	變更
2.0.9	錯誤修正和改善 - 改善在核心裝置上建立和更新 AWS IoT SiteWise 資源的可靠性。 - 新增其他本機 API 操作，可用來監控核心裝置上安裝的元件、每個元件的版本，以及每個元件的狀態。您可以在核心裝置上 AWS OpsHub 應用程式 AWS IoT SiteWise 的設定索引標籤上檢視此資訊。 - 新增此元件執行之 Docker 容器的運作狀態。您可以執行 <code>docker ps</code> 命令來檢視容器的運作狀態。
2.0.7	錯誤修正和改善 修正在核心裝置上檢視 AWS IoT SiteWise Monitor 入口網站的支援。
2.0.6	錯誤修正和改善 修正此元件在核心裝置上運算的 AWS IoT SiteWise <code>statetime()</code>、<code>earliest()</code>、和 <code>latest()</code> 函數。
2.0.5	錯誤修正和改善 - 在轉換中新增對 函數的 AWS IoT SiteWise <code>pretrigger()</code> 支援，此元件會在核心裝置上運算。 - 變更此元件存放輕量型目錄存取通訊協定 (LDAP) 組態以進行身分驗證的路徑。
2.0.2	初始版本。

另請參閱

- AWS IoT SiteWise 《使用者指南》中的 [什麼是 AWS IoT SiteWise ?](#)。

發行者支援的元件

發行者支援的元件位於 的預覽版本中，AWS IoT Greengrass 可能會有所變更。不支援這些元件 AWS。如果每個元件有任何問題，您必須聯絡發佈者。

Greengrass Publisher 支援的元件是由第三方元件供應商開發、提供和服務。第三方元件廠商來自 AWS Partner Device Catalog、AWS Heroes 或社群廠商。您可以直接聯絡第三方元件供應商，購買此目錄中的元件。

Greengrass Publisher 支援的元件包括下列項目：

主題

- [AIShield.Edge](#)
- [AI EdgeLabs 感應器](#)
- [Greengrass S3 擷取器](#)

AIShield.Edge

此元件由 Bosch 提供支援的 AIShield 開發和支援。使用 AIShield.Edge. 此元件旨在無縫部署威脅資訊、量身打造的防禦到邊緣裝置，以保護您的裝置免受 AI 攻擊。

此元件提供下列優點：

- 使用 AIShield AI Security 無縫地從漏洞分析轉換到 中的強化邊緣防禦 AWS
- 輕鬆跨多個邊緣裝置部署量身打造的防禦
- 為支援各種模型類型和架構的各種 AI 設定量身打造的廣泛保護
- 透過與 Amazon SageMaker AI 和 Greengrass 工作流程的無縫整合，隨時保持更新
- 取得潛在威脅的即時洞見，並將資料直接轉送至 AWS IoT Core
- 從 AWS Marketplace 的 AIShield AI Security 進行邊緣防禦部署的凝聚性 AI 安全途徑

此元件必須在下列平台上執行：

- os : linux

如果您有興趣購買此元件，請聯絡 Bosch 軟體和數位解決方案：<AIShield.Contact@bosch.com>。

AI EdgeLabs 感應器

此元件已開發且受 AI EdgeLabs 支援。AI EdgeLabs 感應器是容器型應用程式，其中包含 AI 型威脅偵測和預防功能。AI 感應器會包裝在 Greengrass 元件中，並與其他 Greengrass 元件一起部署為核心裝置上的獨立容器。

此目前元件是容器型代理程式，會持續驗證網路通訊，在 Edge Host 或 IoT 閘道上執行的軟體中尋找威脅模式。此元件使用 eBPF、程序頻寬的行為驗證，以及以主機為基礎的組態。此元件的主要功能是以 NDR/IPS 和 EDR 函數為基礎。

此元件提供下列優點：

- 針對網路攻擊和惡意軟體的 AI 型威脅偵測 (EDR/NDR)
- 自動化 AI 型事件回應 (IPS)
- 主機本機威脅情報，在外部將資料傳輸降至最低
- 使用 Docker 和 Greengrass 進行輕量部署

此元件必須在下列其中一個平台上執行：

- os : linux

如果您有興趣購買此元件，請聯絡 AI EdgeLabs : <contact@edgelabs.ai>。

Greengrass S3 擷取器

此元件是由 Nathan Glover 開發和支援。Greengrass S3 Ingestor 元件設計為與[串流管理員元件](#)搭配使用。此元件會從串流管理員取得以行分隔的 JSON 訊息串流，並將其批次處理為 GZIP 檔案。此元件可有效將資料擷取至 Amazon S3 以進行進一步處理或儲存。此元件不支援 AWS 雲端 即時將資料傳送至。

此元件必須在下列其中一個平台上執行：

- os : linux
- os : Windows

如果您有興趣購買此元件，請聯絡 Nathan Glover : <nathan@glovers.id.au>。

社群元件

Greengrass 軟體目錄是由 Greengrass 社群開發的 Greengrass 元件索引。您可以從此目錄中下載、修改和部署元件，以建立 Greengrass 應用程式。您可以在以下連結檢視目錄：<https://github.com/aws-greengrass/aws-greengrass-software-catalog>。

每個元件都有一個您可以探索的公有 GitHub 儲存庫。在 GitHub 上檢視 Greengrass 軟體目錄，以尋找社群元件的完整清單。例如，此目錄包含下列元件：

- [Amazon Kinesis Video Streams](#)

此元件會從使用[即時串流通訊協定 \(RTSP\)](#) 的本機攝影機擷取音訊和視訊串流。然後，元件會將音訊和影片串流上傳至 [Amazon Kinesis Video Streams](#)。

- [藍牙 IoT 閘道](#)

此元件使用 [BluePy](#) 程式庫，可讓與低功耗藍牙 (LE) 裝置進行通訊，以建立藍牙 LE 用戶端介面。

- [憑證輪換器](#)

此元件可讓您在機群之間大規模輪換 AWS IoT Greengrass 核心裝置憑證和私有金鑰。

- [容器化安全通道](#)

此元件提供 Docker 容器，用於安全通道，其具有可重複使用配方中的所有相依性和相符的程式庫，而這些配方不依賴特定的主機作業系統。

- [格拉法納](#)

此元件可讓您在 Greengrass 核心裝置上託管 [Grafana](#) 伺服器。您可以使用 Grafana 儀表板來視覺化和管理核心裝置上的資料。

- [GStreamer for Amazon Lookout for Vision](#)

此元件提供 GStreamer 外掛程式，讓您可以在自訂 GStreamer 管道中執行 Lookout for Vision 異常偵測。

- [家庭助理](#)

此元件可讓客戶使用 [Home Assistant](#) 來提供智慧家庭裝置的本機控制。它提供與邊緣和雲端 AWS 服務整合，以提供擴展 Home Assistant 的家庭自動化解決方案。

- [InfluxDBGrafana 儀表板](#)

此元件提供一鍵式體驗來設定 InfluxDB 和 Grafana 元件。它將 InfluxDB 連接到 Grafana，並自動設定本機 Grafana 儀表板，以即時轉譯 AWS IoT Greengrass 遙測。

- [InfluxDB](#)

此元件在 Greengrass 核心裝置上提供 [InfluxDB](#) 時間序列資料庫。您可以使用此元件來處理來自 IoT 感應器的資料、即時分析資料，以及監控邊緣的操作。

- [InfluxDB 發佈者](#)

此元件會將 AWS IoT Greengrass 系統運作狀態遙測從 [Nucleus 發射器外掛程式](#) 轉送至 InfluxDB。此元件也可以將自訂遙測轉送至 InfluxDB。

- [IoT pubsub 架構](#)

此架構提供應用程式架構、範本程式碼和可部署的範例，可協助改善使用 AWS IoT Greengrass v2 自訂元件的分散式事件驅動 IoT pubsub 應用程式的程式碼品質。如需詳細資訊，請參閱[建立 AWS IoT Greengrass 元件](#)。

- [Jupyter 實驗室](#)

此元件會將 JupyterLab 部署到 AWS IoT Greengrass 核心裝置。Jupyter 環境可存取由設定的程序和環境變數資源 AWS IoT Greengrass，簡化測試和開發以 Python 撰寫元件的程序。

- [本機 Web 伺服器](#)

此元件可讓您在 Greengrass 核心裝置上建立本機 Web 使用者介面。例如，您可以建立本機 Web 使用者介面，讓您設定裝置和應用程式設定或監控裝置。

- [LoRaWaN 通訊協定轉接器](#)

此元件會從使用 LoRaWaN 通訊協定的本機無線裝置擷取資料，這是低功耗廣域網路 (LPWAN) 通訊協定。元件可讓您在本地分析資料並對其採取行動，而無需與雲端通訊。

- [Modbus TCP](#)

此元件會使用 ModbusTCP 通訊協定從本機裝置收集資料，並將其發佈至選取的資料串流。

- [Node-RED](#)

此元件會使用 NPM 在 AWS IoT Greengrass 核心裝置上安裝 Node-RED。元件取決於必須明確部署和設定的 [Node-RED Auth](#) 元件。您可以使用 [Node-RED CLI for Greengrass](#) 將 Node-RED 流程部署至 AWS IoT Greengrass 裝置。

- [Node-RED Docker](#)

此元件會使用官方 Node-RED Docker 容器在 AWS IoT Greengrass 核心裝置上安裝 Node-RED。元件取決於必須明確部署和設定的 [Node-RED Auth](#) 元件。您可以使用 [Node-RED CLI for Greengrass](#) 將 Node-RED 流程部署至 AWS IoT Greengrass 裝置。

- [Node-RED 驗證](#)

此元件會設定使用者名稱和密碼，以保護 AWS IoT Greengrass 核心裝置上執行的 Node-RED 執行個體。

- [OpenThread 邊界路由器](#)

此元件部署 OpenThread 邊界路由器 Docker 容器。元件有助於編寫包含 Thread 邊界路由器的 Matter 裝置。

- [OSI Pi 串流資料連接器](#)

此元件提供從 OSI Pi Data Archive 到現代資料架構的串流即時資料擷取 AWS。它整合到透過 AWS IoT PubSub 訊息集中管理的 OSI Pi 資產架構。

- [Parsec 提供者](#)

此元件可讓 AWS IoT Greengrass 裝置使用[來自雲端原生運算基金會 \(CNCF\)](#) 的開放原始碼 [Parsec](#) 專案來整合硬體安全解決方案。

- [PostgreSQL 資料庫](#)

此元件在邊緣提供 [PostgreSQL](#) 關聯式資料庫的支援。客戶可以使用此元件在 Docker 容器內佈建和管理本機 PostgreSQL 執行個體。

- [S3 檔案上傳器](#)

此元件會監控新檔案的目錄、將其上傳至 Amazon Simple Storage Service (Amazon S3)，然後在成功上傳後將其刪除。

- [Secrets Manager 用戶端](#)

此元件提供 CLI 工具，可供其他需要從配方生命週期指令碼中的 Secrets Manager 元件擷取秘密的元件使用。

- [TES 路由至容器](#)

此元件會在 AWS IoT Greengrass 裝置上設定可轉換或可轉換，以便將[字符交換服務](#)元件與容器搭配使用。

- [WebRTC](#)

此元件會從連接至 AWS IoT Greengrass 核心裝置的 RTSP 攝影機擷取音訊和視訊串流。然後，元件會將音訊和視訊串流轉換為peer-to-peer通訊，或透過 Amazon Kinesis Video Streams 轉送。

若要請求功能或報告錯誤，請在該元件的儲存庫上開啟 GitHub 問題。AWS 不支援社群元件。如需詳細資訊，請參閱每個元件儲存庫中的 CONTRIBUTING.md 檔案。

數個 AWS 提供的元件也是開放原始碼。如需詳細資訊，請參閱[開放原始碼 AWS IoT Greengrass 核心軟體](#)。

AWS IoT Greengrass 開發工具

使用 AWS IoT Greengrass 開發工具來建立、測試、建置、發佈和部署自訂 Greengrass 元件。

- [Greengrass 開發套件 CLI](#)

在本機開發環境中使用 AWS IoT Greengrass 開發套件命令列界面 (GDK CLI)，從 [Greengrass 軟體目錄](#) 中的範本和社群元件建立元件。您可以使用 GDK CLI 建置元件，並將元件以私有元件的形式發佈至 AWS IoT Greengrass 服務 AWS 帳戶。

- [Greengrass 命令列界面](#)

在 Greengrass 核心裝置上使用 Greengrass Command Line Interface (Greengrass CLI) 來部署和偵錯 Greengrass 元件。Greengrass CLI 是一種元件，您可以部署到核心裝置，以建立本機部署、檢視已安裝元件的詳細資訊，以及探索日誌檔案。

- [本機偵錯主控台](#)

使用 Greengrass 核心裝置上的本機偵錯主控台，使用本機儀表板 Web 介面來部署和偵錯 Greengrass 元件。本機偵錯主控台是您部署到核心裝置的元件，以建立本機部署並檢視已安裝元件的詳細資訊。

AWS IoT Greengrass 也提供下列 SDKs，您可以在自訂 Greengrass 元件中使用：

- AWS IoT Device SDK，其中包含程序間通訊 (IPC) 程式庫。如需詳細資訊，請參閱[使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#)。
- 串流管理員 SDK，可用來將資料串流傳輸至 AWS 雲端。如需詳細資訊，請參閱[管理 Greengrass 核心裝置上的資料串流](#)。

主題

- [AWS IoT Greengrass 開發套件命令列介面](#)
- [Greengrass 命令列界面](#)
- [使用 AWS IoT Greengrass 測試架構](#)

AWS IoT Greengrass 開發套件命令列介面

AWS IoT Greengrass 開發套件命令列界面 (GDK CLI) 提供可協助您開發[自訂 Greengrass 元件](#)的功能。您可以使用 GDK CLI 來建立、建置和發佈自訂元件。當您使用 GDK CLI 建立元件儲存庫時，

您可以從範本或 [Greengrass 軟體目錄](#) 中的社群元件開始。然後，您可以選擇建置系統，將檔案封裝為 ZIP 封存檔、使用 Maven 或 Gradle 建置指令碼，或執行自訂建置命令。建立元件後，您可以使用 GDK CLI 將其發佈至 AWS IoT Greengrass 服務，以便您可以使用 AWS IoT Greengrass 主控台或 API 將元件部署至 Greengrass 核心裝置。

當您在沒有 GDK CLI 的情況下開發 Greengrass 元件時，每次建立新的元件版本時，都必須更新 [元件配方檔案](#) 中的版本和成品 URIs。當您使用 GDK CLI 時，它可以在每次發佈新版本的元件時自動為您更新版本和成品 URIs。

GDK CLI 是開放原始碼，可在 GitHub 上使用。您可以自訂和擴展 GDK CLI，以滿足元件開發需求。我們邀請您在 GitHub 儲存庫上開啟問題並提取請求。您可以在以下連結找到 GDK CLI 來源：
https : <https://github.com/aws-greengrass/aws-greengrass-gdk-cli>。

先決條件

若要安裝和使用 Greengrass 開發套件 CLI，您需要下列項目：

- AWS 帳戶：如果您沒有帳戶，請參閱 [設定 AWS 帳戶](#)。
- 具有網際網路連線的 Windows、macOS 或類似 Unix 的開發電腦。
- 對於 GDK CLI 1.1.0 版或更新版本，[Python](#) 3.6 或更新版本會安裝在開發電腦上。

對於 GDK CLI 1.0.0 版，[Python](#) 3.8 或更新版本安裝在開發電腦上。

- 安裝在開發電腦上的 [Git](#)。
- AWS Command Line Interface (AWS CLI) 在開發電腦上安裝並設定憑證。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的 [安裝、更新和解除安裝 AWS CLI](#) 和 [設定 AWS CLI](#)。

Note

如果您使用 Raspberry Pi 或其他 32 位元 ARM 裝置，則 install AWS CLI V1。AWS CLI V2 不適用於 32 位元 ARM 裝置。如需詳細資訊，請參閱 [安裝、更新和解除安裝 AWS CLI 版本 1](#)。

- 若要使用 GDK CLI 將元件發佈至 AWS IoT Greengrass 服務，您必須具有下列許可：
 - s3:CreateBucket
 - s3:GetBucketLocation
 - s3:PutObject

- `greengrass:CreateComponentVersion`
- `greengrass:ListComponentVersions`
- 若要使用 GDK CLI 來建置其成品存在於 S3 儲存貯體而非本機檔案系統中的元件，您必須具有下列許可：
 - `s3:ListBucket`

此功能適用於 GDK CLI 1.1.0 版及更新版本。

變更記錄

下表說明每個 GDK CLI 版本中的變更。如需詳細資訊，請參閱 GitHub 上的 [GDK CLI 版本頁面](#)。

版本	變更
1.6.2	錯誤修正和改善 • 修正 Windows gradlew.bat 因相對路徑而無法運作的問題。 • 記錄、測試和封裝的小幅度改進。
1.6.1	錯誤修正和改善 • 新增 CLI 引數剖析的安全修正。 • 讓 GDK 取得最新的 Greengrass 測試架構 (GTF) 版本名稱做為預設 GTF 版本。 • 可讓 GDK 使用客戶更新至最新版本的舊版 GTF 來建議客戶。
1.6.0	新功能 • 在 <code>component build</code> 和 <code>component publish</code> 命令期間，針對 Greengrass 配方結構描述新增配方驗證檢查。此更新可協助開發人員在元件建立程序中稍早在其元件配方中識別可採取動作的問題。 • 將可信度測試套件新增至範本，該範本可由 <code>test-e2e init</code> 命令下拉。此可信度測試套件包含八項一般測試，可用於並擴充以滿足基本元件測試需求。 錯誤修正和改善 • 將 <code>test-e2e</code> 命令使用的預設 Greengrass 測試架構 (GTF) 版本更新為 1.2.0 版。

版本	變更
1.5.0	錯誤修正和改善 當 <code>build_system</code> 為 <code>zip</code> 時，更新 <code>excludes</code> 建置選項所辨識的模式 <code>zip</code>。此版本現在會根據其萬用字元來辨識符合路徑名稱的 <code>glob</code> 模式。這會啟用要排除哪些目錄的自訂規格。
1.4.0	新功能 - 新增新 <code>config</code> 命令，以啟動互動式提示來修改現有 GDK 組態檔案中的欄位。 - 修改 <code>gdk component build</code> 和 <code>gdk component publish</code> 命令，以確認配方大小在 Greengrass 要求 (≤ 16000 位元組) 內，然後再繼續。 錯誤修正和改善 - 當配方語法錯誤導致建置無法完成以保持意識時，會在 <code>gdk component build</code> 命令的輸出中新增額外的記錄。 - 由於 Open Test Framework 已重新命名 <code>otf-options</code> 和 <code>otf-version</code> 為 Greengrass Testing Framework，因此將 <code>gtf-options</code> 和 <code>gtf-version</code> 分別重新命名為 <code>otf-options</code> 和 <code>otf-version</code>。
1.3.0	新功能 - 新增 <code>test-e2e</code> 命令，以支援使用 Open Test Framework 進行元件的 end-to-end 測試。 - 新增新的組態選項 <code>zip_name</code>，以支援具有 <code>zip</code> 組建系統的可設定 <code>zip</code> 檔案名稱。 - 將 GDK 組態檔案中的 <code>region</code> 屬性設為選用。 錯誤修正和改善 - 修正即使使用 <code>--name</code> 引數初始化 GDK 專案時不存在指定的範本或儲存庫，也會建立新目錄的問題。
1.2.3	錯誤修正和改善 - 修正儲存貯體建立因錯誤處理而失敗的問題。 - 修正元件配方中清單結構移除的問題。

版本	變更
1.2.2	錯誤修正和改善 • 配方索引鍵不再區分大小寫。 • 新增檢查，以判斷儲存貯體是否存在於 中，AWS 區域 且使用者可以在建立新儲存貯體之前存取。要求使用者擁有 <code>GetBucketLocation</code> 許可。 • 修正 GDK CLI 組態檔案中 <code>excludes</code> 關鍵字的問題。
1.2.1	錯誤修正和改善 • 接受 <code>gdk-config.json</code> 檔案中區域組態項目 AWS 區域 中的加拿大 (中部) (<code>ca-central-1</code>)。 • 將 <code>--region</code> GDK CLI 引數的問題修正為 <code>publish</code> 命令。
1.2.0	新功能 • 將 <code>options</code> 項目新增至 GDK CLI <code>build</code> 組態檔案中的組態。支援 <code>excludes options</code> 在使用 <code>zip</code> 建置系統時，從 <code>zip</code> 成品中排除特定檔案。 • 新增 <code>gradlew</code> 建置系統以使用 Gradle Wrapper 來建置元件。 • 為建置選項新增對 Kotlin DSL <code>gradle</code> 組建檔案的支援。 • 將 <code>options</code> 項目新增至 GDK CLI <code>publish</code> 組態檔案中的組態。支援 <code>file_upload_args</code> 下方 <code>options</code>，以在將檔案上傳至 Amazon S3 時提供額外的引數。 錯誤修正和改善 • 修正執行建置命令之前 Gradle 組建未清除的問題。 • 修正建置命令失敗時，組建未結束的問題。 • 改善 <code>gdk component list</code> 命令的輸出格式。

版本	變更
1.1.0	新功能 • 新增對 Gradle 建置系統 的支援。 • 在 Windows 裝置上新增對 Maven 建置系統 的支援。 • 將 <code>--bucket</code> 引數新增至 元件發佈 命令。您可以使用此引數指定 GDK CLI 上傳元件成品的確切儲存貯體。 • 將 <code>--name</code> 引數新增至 元件 init 命令。您可以使用此選項來指定 GDK CLI 初始化元件的資料夾。 • 新增對 S3 儲存貯體中存在但不在本機元件建置資料夾中之元件成品的支援。您可以使用此功能來降低大型元件成品的頻寬成本，例如機器學習模型。 錯誤修正和改善 • 更新 元件發佈 命令，以檢查元件是否已在發佈元件之前建置。如果未建置元件，此命令現在會為您 建置元件。 • 修正當 ZIP 檔案名稱包含大寫字母時，zip 組建系統無法在 Windows 裝置上組建的問題。 • 改善日誌訊息格式，並在執行 Python 3.8 版之前的 INFO 裝置上將預設日誌層級變更為。 • 將最低 Python 版本需求變更為 Python 3.6。
1.0.0	初始版本。

安裝或更新 AWS IoT Greengrass 開發套件命令列界面

AWS IoT Greengrass 開發套件命令列界面 (GDK CLI) 是以 Python 為基礎，因此您可以使用 pip 將其安裝在開發電腦上。

Tip

您也可以 Python 虛擬環境中安裝 GDK CLI，例如 [venv](#)。如需詳細資訊，請參閱 Python 3 文件中的 [虛擬環境和套件](#)。

安裝或更新 GDK CLI

1. 執行下列命令，從其 [GitHub 儲存庫](#) 安裝最新版本的 GDK CLI。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

Note

若要安裝特定版本的 GDK CLI，請將 *versionTag* 取代為要安裝的版本標籤。您可以在其 [GitHub 儲存庫](#) 中檢視 GDK CLI 的版本標籤。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
```

2. 執行下列命令來驗證 GDK CLI 是否已成功安裝。

```
gdk --help
```

如果找不到 gdk 命令，請將其資料夾新增至 PATH。

- 在 Linux 裝置上，將 `/home/MyUser/.local/bin` 新增至 PATH，並以您的使用者名稱取代 *MyUser*。
- 在 Windows 裝置上，將 `PythonPath\Scripts` 新增至 PATH，並將 *PythonPath* 取代為您裝置上的 Python 資料夾路徑。

您現在可以使用 GDK CLI 來建立、建置和發佈 Greengrass 元件。如需如何使用 GDK CLI 的詳細資訊，請參閱 [AWS IoT Greengrass 開發套件命令列界面命令](#)。

AWS IoT Greengrass 開發套件命令列界面命令

Development AWS IoT Greengrass Kit Command-Line Interface (GDK CLI) 提供命令列介面，可用於在開發電腦上建立、建置和發佈 Greengrass 元件。GDK CLI 命令使用以下格式。

```
gdk <command> <subcommand> [arguments]
```

[安裝 GDK CLI](#) 時，安裝程式會 gdk 新增至 PATH，以便您可以從命令列執行 GDK CLI。

您可以搭配任何命令使用下列引數：

- 如需 GDK CLI 命令的相關資訊，`--help`請使用 `-h`或。
- 使用 `-v`或 `--version` 來查看已安裝的 GDK CLI 版本。
- 使用 `-d`或 `--debug`輸出詳細日誌，您可以用來偵錯 GDK CLI。

本節說明 GDK CLI 命令，並提供每個命令的範例。每個命令的摘要會顯示其引數及其用量。選用引數以方括號顯示。

可用命令

- [元件](#)
- [config](#)
- [test-e2e](#)

元件

使用 AWS IoT Greengrass 開發套件命令列界面 (GDK CLI) 中的 `component` 命令來建立、建置和發佈自訂 Greengrass 元件。

子命令

- [init](#)
- [build](#)
- [發布](#)
- [列出](#)

init

從元件範本或社群元件初始化 Greengrass 元件資料夾。

GDK CLI 從 [Greengrass 軟體目錄](#)擷取社群元件，並從 [AWS IoT Greengrass GitHub 上的元件範本儲存庫](#)擷取元件範本。

Note

如果您使用 GDK CLI 1.0.0 版，您必須在空的資料夾中執行此命令。GDK CLI 會將範本或社群元件下載到目前的資料夾。

如果您使用 GDK CLI 1.1.0 版或更新版本，您可以指定 `--name` 引數來指定 GDK CLI 下載範本或社群元件的資料夾。如果您使用此引數，請指定不存在的資料夾。GDK CLI 會為您建立資料夾。如果您未指定此引數，GDK CLI 會使用目前的資料夾，該資料夾必須是空的。

如果元件使用 [zip 組建系統](#)，GDK CLI 會將元件資料夾中的特定檔案壓縮為與元件資料夾名稱相同的 zip 檔案。例如，如果元件資料夾的名稱是 `HelloWorld`，GDK CLI 會建立名為 `HelloWorld.zip` 的 zip 檔案。在元件配方中，zip 成品名稱必須符合元件資料夾的名稱。如果您在 Windows 裝置上使用 GDK CLI 1.0.0 版，元件資料夾和 zip 檔案名稱只能包含小寫字母。

如果您將範本或社群元件初始化為使用 zip build 系統的資料夾，其名稱與範本或元件不同，則必須在元件配方中變更 zip 成品名稱。更新 Artifacts 和 Lifecycle 定義，讓 zip 檔案名稱符合元件資料夾的名稱。下列範例會反白顯示 Artifacts 和 Lifecycle 定義中的 zip 檔案名稱。

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}"
```

概要

```
$ gdk component init
  [--language]
  [--template]
  [--repository]
  [--name]
```

引數 (從元件範本初始化)

- -l, --language – 用於您指定範本的程式設計語言。

您必須指定 --repository 或 --language 和 --template。

- -t, --template – 用於本機元件專案的元件範本。若要檢視可用的範本，請使用 [清單](#) 命令。

您必須指定 --repository 或 --language 和 --template。

- -n, --name – (選用) GDK CLI 初始化元件的本機資料夾名稱。指定不存在的資料夾。GDK CLI 會為您建立 資料夾。

此功能適用於 GDK CLI 1.1.0 版及更新版本。

引數 (從社群元件初始化)

- -r, --repository – 要簽出至本機資料夾的社群元件。若要檢視可用的社群元件，請使用 [list](#) 命令。

您必須指定 --repository 或 --language 和 --template。

- -n, --name – (選用) GDK CLI 初始化元件的本機資料夾名稱。指定不存在的資料夾。GDK CLI 會為您建立 資料夾。

此功能適用於 GDK CLI 1.1.0 版及更新版本。

輸出

下列範例顯示執行此命令以初始化 Python Hello World 範本中的元件資料夾時產生的輸出。

```
$ 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.
```

下列範例顯示執行此命令從社群元件初始化元件資料夾時產生的輸出。

```
$ 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

將元件的來源建置到您可以發佈到服務的配方和成品中 AWS IoT Greengrass。GDK CLI 會執行您在 [GDK CLI 組態檔案](#) 中指定的建置系統gdk-config.json。您必須在gdk-config.json檔案存在的相同資料夾中執行此命令。

當您執行此命令時，GDK CLI 會在元件greengrass-build資料夾中的 資料夾中建立配方和成品。GDK CLI 會將配方儲存在 greengrass-build/recipes 資料夾中，並將成品儲存在 greengrass-build/artifacts/*componentName/componentVersion* 資料夾中。

如果您使用 GDK CLI 1.1.0 版或更新版本，元件配方可以指定 S3 儲存貯體中存在的成品，但不能指定本機元件建置資料夾中存在的成品。當您開發具有大型成品的元件時，您可以使用此功能來減少頻寬用量，例如機器學習模型。

建置元件之後，您可以執行下列其中一項操作，在 Greengrass 核心裝置上進行測試：

- 如果您在與執行 AWS IoT Greengrass Core 軟體不同的裝置上開發，則必須發佈元件，將其部署到 Greengrass 核心裝置。將元件發佈至 AWS IoT Greengrass 服務，並將其部署至 Greengrass 核心裝置。如需詳細資訊，請參閱[發佈](#)命令和 [建立部署](#)。
- 如果您在執行 AWS IoT Greengrass Core 軟體的相同裝置上開發，您可以將元件發佈至要部署 AWS IoT Greengrass 的服務，或者您可以建立本機部署來安裝和執行元件。若要建立本機部署，請使用 Greengrass CLI。如需詳細資訊，請參閱 [Greengrass 命令列界面](#) 和 [使用本機部署測試 AWS](#)

[IoT Greengrass 元件](#)。當您建立本機部署時，請將指定greengrass-build/recipes為配方資料夾，並將指定greengrass-build/artifacts為成品資料夾。

概要

```
$ gdk component build
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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'.
```

發布

將此元件發佈至 AWS IoT Greengrass 服務。此命令會將建置成品上傳至 S3 儲存貯體、更新配方中的成品 URI，以及從配方建立新的版本元件。GDK CLI 使用您在 GDK [CLI 組態檔案中](#)指定的 S3 儲存貯體和 AWS 區域gdk-config.json。您必須在gdk-config.json檔案存在的相同資料夾中執行此命令。

如果您使用 GDK CLI 1.1.0 版或更新版本，您可以指定--bucket引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳到名為的 S3 儲存貯

體 `bucket-region-accountId`，其中儲存#體和##是您在 `gdk-config.json` 中指定的值，而 `accountId` 是您的 AWS 帳戶 ID。如果儲存貯體不存在，GDK CLI 會建立儲存貯體。

如果您使用 GDK CLI 1.2.0 版或更新版本，您可以使用 `--region` 參數覆寫 GDK CLI 組態檔案中 AWS 區域 指定的。您也可以使用 `--options` 參數指定其他選項。如需可用選項的清單，請參閱 [Greengrass 開發套件 CLI 組態檔案](#)。

當您執行此命令時，GDK CLI 會使用您在配方中指定的版本發佈元件。如果您指定 `NEXT_PATCH`，GDK CLI 會使用不存在的下一個修補程式版本。語意版本使用 `major.minor.patch` 編號系統。如需詳細資訊，請參閱 [語意版本規格](#)。

Note

如果您使用 GDK CLI 1.1.0 版或更新版本，當您執行此命令時，GDK CLI 會檢查元件是否已建置。如果未建置元件，則 GDK CLI [會在發佈元件之前建置](#) 元件。

概要

```
$ gdk component publish  
  [--bucket] [--region] [--options]
```

Arguments (引數)

- `-b`，`--bucket` – (選用) 指定 GDK CLI 發佈元件成品的 S3 儲存貯體名稱。

如果您未指定此引數，GDK CLI 會上傳到名為 `bucket-region-accountId` 的 S3 儲存貯體，其中儲存#體和##是您在 `gdk-config.json` 中指定的值，而 `accountId` 是您的 AWS 帳戶 ID。如果儲存貯體不存在，GDK CLI 會建立儲存貯體。

如果儲存貯體不存在，GDK CLI 會建立儲存貯體。

此功能適用於 GDK CLI 1.1.0 版及更新版本。

- `-r`，`--region` – (選用) 建立元件時，將的名稱指定 AWS 區域 為。此引數會覆寫 GDK CLI 組態中的區域名稱。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

- `-o`，`--options` (選用) 指定發佈元件的選項清單。引數必須是有效的 JSON 字串或包含發佈選項的 JSON 檔案路徑。此引數會覆寫 GDK CLI 組態中的選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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'.
```

列出

擷取可用元件範本和社群元件的清單。

GDK CLI 從 [Greengrass 軟體目錄](#)擷取社群元件，並從 [AWS IoT Greengrass GitHub 上的元件範本儲存庫](#)擷取元件範本。

您可以將此命令的輸出傳遞至 [init](#) 命令，以從範本和社群元件初始化元件儲存庫。

概要

```
$ gdk component list
[--template]
```

```
[--repository]
```

Arguments (引數)

- `-t` , `--template` – (選用) 指定此引數以列出可用的元件範本。此命令會以格式輸出每個範本的名稱和語言 *name-language*。例如，在中 HelloWorld-python，範本名稱為 HelloWorld，語言為 python。
- `-r` , `--repository` – (選用) 指定此引數以列出可用的社群元件儲存庫。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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

使用 AWS IoT Greengrass 開發套件命令列界面 (GDK CLI) 中的 `config` 命令來修改組態檔案中的 GDK 組態 `gdk-config.json`。

子命令

- [update](#)

update

啟動互動式提示來修改現有 GDK 組態檔案中的欄位。

概要

```
$ gdk config update
[--component]
```

Arguments (引數)

- `-c` , `--component` – 更新 `gdk-config.json` 檔案中與元件相關的欄位。此引數是必要的，因為它是唯一的選項。

輸出

下列範例顯示執行此命令以設定元件時產生的輸出。

```
$ 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

使用 AWS IoT Greengrass 開發套件命令列界面 (GDK CLI) 中的 `test-e2e` 命令，在 GDK 專案中初始化、建置和執行 end-to-end 測試模組。

子命令

- [init](#)
- [build](#)
- [run](#)

init

使用使用 Greengrass 測試架構 (GTF) 的測試模組初始化現有的 GDK CLI 專案。

根據預設，GDK CLI 會從 [AWS IoT Greengrass GitHub 上的元件範本儲存庫擷取數學模組範本](#)。此 maven 模組隨附 `aws-greengrass-testing-standalone` JAR 檔案的相依性。

此命令會在 GDK 專案 `gg-e2e-tests` 內建立新的目錄，稱為 `test-e2e`。如果測試模組目錄已存在且不是空的，則命令會結束而不執行任何動作。此 `gg-e2e-tests` 資料夾包含在 maven 專案中建構的 Cucumber 功能和步驟定義。

根據預設，此命令會嘗試使用最新版本的 GTF。

概要

```
$ gdk test-e2e init
  [--gtf-version]
```

Arguments (引數)

- `-ov`、`--gtf-version-` (選用) GTF 的版本，用於 GDK 專案中的end-to-end測試模組。此值必須是版本中的[其中一個 GTF](#) 版本。此引數會覆寫 GDK CLI 組態`gtf_version`中的。

輸出

下列範例顯示執行此命令以使用測試模組初始化 GDK 專案時產生的輸出。

```
$ 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

您必須先執行 `來建置元件`，`gdk component build`end-to-end測試模組。

建置end-to-end測試模組。GDK CLI 會使用您在 `test-e2e` 屬性下的 [GDK CLI 組態檔案中](#)指定的建置系統`gdk-config.json`來建置測試模組。您必須在`gdk-config.json`檔案存在的相同資料夾中執行此命令。

根據預設，GDK CLI 會使用 `maven` 建置系統來建置測試模組。執行 `gdk test-e2e build`命令需要 [Maven](#)。

如果測試功能檔案具有 `GDK_COMPONENT_NAME`和 `等變數GDK_COMPONENT_RECIPE_FILE`來插補，則必須先執行 `來建置元件`，`gdk-component-build`才能建置測試模組。

當您執行此命令時，GDK CLI 會從 GDK 專案組態插入所有變數，並建置`gg-e2e-tests`模組以產生最終測試 JAR 檔案。

概要

```
$ gdk test-e2e build
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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'
.....
```

run

使用 GDK 組態檔案中的測試選項執行測試模組。

Note

您必須先執行 `gdk test-e2e buildend-to-end` 測試。

概要

```
$ gdk test-e2e run
[--gtf-options]
```

Arguments (引數)

- `-oo` , `--gtf-options` – (選用) 指定執行end-to-end測試的選項清單。引數必須是有效的 JSON 字串或包含 GTF 選項的 JSON 檔案路徑。組態檔案中提供的選項會與命令引數中提供的選項合併。如果兩個位置都有 選項，則引數中的一個會比組態檔案中的某個選項有前置。

如果此命令中未指定 `tags` 選項，則 GDK 會使用 `Sample` 進行標籤。如果 `ggc-archive` 未指定，則 GDK 會下載最新版本的 Greengrass 核封存。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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
.....
```

Greengrass 開發套件 CLI 組態檔案

AWS IoT Greengrass Development Kit Command-Line Interface (GDK CLI) 會從名為 `greengrass-gdk-cli` 的組態檔案讀取 `gdk-config.json` 以建置和發佈元件。此組態檔案必須存在於元件儲存庫的根目錄中。您可以使用 GDK CLI [init 命令](#) 來使用此組態檔案初始化元件儲存庫。

主題

- [GDK CLI 組態檔案格式](#)
- [GDK CLI 組態檔案範例](#)

GDK CLI 組態檔案格式

當您定義元件的 GDK CLI 組態檔案時，請以 JSON 格式指定下列資訊。

`gdk_version`

與此元件相容的 GDK CLI 最低版本。此值必須是 [版本的](#) 其中一個 GDK CLI 版本。

`component`

此元件的組態。

componentName

author

元件的作者或發佈者。

version

元件的版本。請指定下列其中一項：

- **NEXT_PATCH** – 選擇此選項時，GDK CLI 會在您發佈元件時設定版本。GDK CLI 會查詢 AWS IoT Greengrass 服務，以識別元件的最新發佈版本。然後，它會將版本設定為該版本之後的下一個修補程式版本。如果您先前尚未發佈元件，GDK CLI 會使用版本 **1.0.0**。

如果您選擇此選項，則無法使用 [Greengrass CLI](#) 將元件本機部署和測試到執行 AWS IoT Greengrass Core 軟體的本機開發電腦。若要啟用本機部署，您必須改為指定語意版本。

- 語意版本，例如 **1.0.0**。語意版本使用 major.minor.patch 編號系統。如需詳細資訊，請參閱[語意版本規格](#)。

如果您在想要部署和測試元件的 Greengrass 核心裝置上開發元件，請選擇此選項。您必須使用特定版本建置元件，才能使用 [Greengrass CLI](#) 建立本機部署。

build

用來將此元件的來源建置為成品的組態。此物件包含下列資訊：

build_system

要使用的建置系統。您可以從以下選項中選擇：

- **zip** – 將元件的資料夾封裝為 ZIP 檔案，以定義為元件的唯一成品。針對下列類型的元件選擇此選項：
 - 使用解譯程式設計語言的元件，例如 Python 或 JavaScript。
 - 封裝程式碼以外檔案的元件，例如機器學習模型或其他資源。

GDK CLI 會將元件的資料夾壓縮成與元件資料夾名稱相同的 zip 檔案。例如，如果元件資料夾的名稱為 HelloWorld，GDK CLI 會建立名為的 zip 檔案HelloWorld.zip。

Note

如果您在 Windows 裝置上使用 GDK CLI 1.0.0 版，元件資料夾和 zip 檔案名稱只能包含小寫字母。

當 GDK CLI 將元件的資料夾壓縮為 zip 檔案時，它會略過下列檔案：

- gdk-config.json 檔案

- 配方檔案 (recipe.json 或 recipe.yaml)
- 建置資料夾，例如 greengrass-build
- maven – 執行 mvn clean package 命令，將元件的來源建置為成品。針對使用 [Maven](#) 的元件選擇此選項，例如 Java 元件。

在 Windows 裝置上，此功能適用於 GDK CLI 1.1.0 版及更新版本。

- gradle – 執行 gradle build 命令，將元件的來源建置為成品。針對使用 [Gradle](#) 的元件選擇此選項。此功能適用於 GDK CLI 1.1.0 版及更新版本。

gradle 建置系統支援 Kotlin DSL 做為建置檔案。此功能適用於 GDK CLI 1.2.0 版及更新版本。

- gradlew – 執行 gradlew 命令，將元件的來源建置為成品。針對使用 [Gradle Wrapper](#) 的元件選擇此選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

- custom – 執行自訂命令，將元件的來源建置到配方和成品中。在 custom_build_command 參數中指定自訂命令。

custom_build_command

(選用) 要為自訂建置系統執行的自訂建置命令。如果您 custom 為指定，則必須指定此參數 build_system。

Important

此命令必須在元件資料夾中的下列資料夾中建立配方和成品。當您執行 [元件建置命令](#) 時，GDK CLI 會為您建立這些資料夾。

- 配方資料夾：greengrass-build/recipes
- 成品資料夾：greengrass-build/artifacts/*componentName*/*componentVersion*

將 *componentName* 取代為元件名稱，並將 *componentVersion* 取代為元件版本 或 NEXT_PATCH。

您可以指定單一字串或字串清單，其中每個字串都是命令中的字詞。例如，若要執行 C++ 元件的自訂建置命令，您可以指定 **cmake --build build --config Release** 或 **["cmake", "--build", "build", "--config", "Release"]**。

若要檢視自訂建置系統的範例，請參閱 [aws.greengrass.labs.LocalWebServer community component GitHub 上的](https://aws.github.io/greengrass-labs-local-web-server-community-component)。

options

(選用) 元件建置程序期間使用的其他組態選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

excludes

定義建置 zip 檔案時要從元件目錄中排除哪些檔案的 glob 模式清單。只有在 build_system 為 zip 時才有效。

Note

在 GDK CLI 1.4.0 版及更早版本中，符合排除清單中項目的任何檔案都會從元件的所有子目錄中排除。若要在 GDK CLI 1.5.0 版和更新版本中實現相同的行為，請在排除清單中 `**/` 的現有項目前面。例如，`*.txt` 只會從目錄排除文字檔案；`**/*.txt` 會從所有目錄和子目錄中排除文字檔案。

在 GDK CLI 1.5.0 版及更新版本中，當 excludes 在 GDK 組態檔案中定義時，您可能會在元件建置期間看到警告。若要停用此警告，請將環境變數 `GDK_EXCLUDES_WARN_IGNORE` 設定為 `true`。

GDK CLI 一律會從 zip 檔案排除下列檔案：

- `gdk-config.json` 檔案
- 配方檔案 (`recipe.json` 或 `recipe.yaml`)
- 建置資料夾，例如 `greengrass-build`

預設會排除下列檔案。不過，您可以使用 excludes 選項來控制哪些檔案遭到排除。

- 任何以字首「測試」開頭的資料夾 (`test*`)
- 所有隱藏的檔案
- `node_modules` 資料夾

如果您指定 excludes 選項，GDK CLI 只會排除您使用 excludes 選項設定的檔案。如果您未指定 excludes 選項，GDK CLI 會排除先前記下的預設檔案和資料夾。

zip_name

在建置過程中建立 zip 成品時要使用的 zip 檔案名稱。只有在 build_system 為時才有效 zip。如果 build_system 是空的，元件名稱會用於 zip 檔案名稱。

publish

用來將此元件發佈至 AWS IoT Greengrass 服務的組態。

如果您使用 GDK CLI 1.1.0 版或更新版本，您可以指定 --bucket 引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳到名為的 S3 儲存貯體 `bucket-region-accountId`，其中儲存#體和##是您在 中指定的值 gdk-config.json，而 `accountId` 是您的 AWS 帳戶 ID。如果儲存貯體不存在，GDK CLI 會建立儲存貯體。

此物件包含下列資訊：

bucket

用來託管元件成品的 S3 儲存貯體名稱。

region

GDK CLI 發佈此元件 AWS 區域的。

如果您使用 GDK CLI 1.3.0 版或更新版本，則此屬性為選用。

options

(選用) 元件版本建立期間使用的其他組態選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

file_upload_args

JSON 結構，其中包含在將檔案上傳至儲存貯體時傳送至 Amazon S3 的引數，例如中繼資料和加密機制。如需允許的引數清單，請參閱 Boto3 文件中的 [S3Transfer](#) 類別。Boto3

test-e2e

(選用) 在元件的 end-to-end 測試期間使用的組態。此功能適用於 GDK CLI 1.3.0 版及更新版本。

build

build_system – 要使用的建置系統。預設選項為 maven。您可以從以下選項中選擇：

- `maven` – 執行 `mvn package` 命令來建置測試模組。選擇此選項來建置使用 [Maven](#) 的測試模組。
- `gradle` – 執行 `gradle build` 命令來建置測試模組。針對使用 [Gradle](#) 的測試模組選擇此選項。

`gtf_version`

(選用) 使用 GTF 初始化 GDK 專案時，做為 end-to-end 測試模組相依性的 Greengrass 測試架構 (GTF) 版本。此值必須是版本中的 [其中一個 GTF](#) 版本。預設為 GTF 1.1.0 版。

`gtf_options`

(選用) 在元件的 end-to-end 測試期間使用的其他組態選項。

下列清單包含您可以搭配 GTF 1.1.0 版使用的選項。

- `additional-plugins` – (選用) 其他小黃瓜外掛程式
- `aws-region` – 以 AWS 服務的特定區域端點為目標。預設為 AWS SDK 探索的內容。
- `credentials-path` – 選用 AWS 的設定檔登入資料路徑。預設為主機環境上探索的登入資料。
- `credentials-path-rotation` – AWS 登入資料的選用輪換持續時間。預設為 15 分鐘或 PT15M。
- `csr-path` – 產生裝置憑證的 CSR 路徑。
- `device-mode` – 正在測試的目標裝置。預設為本機裝置。
- `env-stage` – 以 Greengrass 的部署環境為目標。預設為生產。
- `existing-device-cert-arn` – 您要用作 Greengrass 裝置憑證的現有憑證的生成。
- `feature-path` – 包含其他特徵檔案的檔案或目錄。預設為不使用其他功能檔案。
- `gg-cli-version` – 覆寫 Greengrass CLI 的版本。預設為中找到的值 `ggc.version`。
- `gg-component-bucket` – 存放 Greengrass 元件的現有 Amazon S3 儲存貯體的名稱。
- `gg-component-overrides` – Greengrass 元件覆寫的清單。
- `gg-persist` – 測試執行後要保留的測試元素清單。預設行為是不保留任何內容。接受的值為：`aws.resources`、`installed.software` 和 `generated.files`。
- `gg-runtime` – 影響測試如何與測試資源互動的值清單。這些值會取代 `gg.persist` 參數。如果預設值為空，則假設所有測試資源都由測試案例管理，包括安裝的 Greengrass 執行期。接受的值為：`aws.resources`、`installed.software` 和 `generated.files`。
- `ggc-archive` – 封存 Greengrass 核元件的路徑。

- `ggc-install-root` – 安裝 Greengrass 核元件的目錄。預設為 `test.temp.path` 和 `test run` 資料夾。
- `ggc-log-level` – 設定測試執行的 Greengrass 核日誌層級。預設為「INFO」。
- `ggc-tes-rolename` – Core AWS IoT Greengrass 將擔任以存取 AWS 服務的 IAM 角色。如果指定名稱的角色不存在，則會建立一個和預設存取政策。
- `ggc-trusted-plugins` – 需要新增至 Greengrass 的信任外掛程式路徑（主機上）的逗號分隔清單。若要提供 DUT 本身的路徑，請在路徑前面加上 `'dut:'`。
- `ggc-user-name` – Greengrass 核的 `user: group posixUser` 值。預設為目前登入的使用者名稱。
- `ggc-version` – 覆寫執行中 Greengrass 核元件的版本。預設為 `ggc.archive` 中的值。
- `log-level` – 測試執行的日誌層級。預設為「INFO」。
- `parallel-config` – 批次索引集和批次數量做為 JSON 字串。批次索引的預設值為 0，批次數量為 1。
- `proxy-url` – 設定所有測試以透過此 URL 路由流量。
- `tags` – 僅執行功能標籤。可與 `'&'` 交集
- `test-id-prefix` – 套用至所有測試特定資源的通用字首，包括 AWS 資源名稱和標籤。預設為「gg」字首。
- `test-log-path` – 將包含整個測試執行結果的目錄。預設為「testResults」。
- `test-results-json` – 標記以判斷是否產生寫入磁碟的 Cucumber JSON 報告。預設為 `true`。
- `test-results-log` – 用來判斷主控台輸出是否寫入磁碟的旗標。預設為 `false`。
- `test-results-xml` – 標記以判斷是否產生寫入磁碟的 JUnit XML 報告。預設為 `true`。
- `test-temp-path` – 產生本機測試成品的目錄。預設為字首為 `gg-testing` 的隨機暫存目錄。
- `timeout-multiplier` – 提供給所有測試逾時的乘數。預設值為 1.0。

GDk CLI 組態檔案範例

您可以參考下列 GDk CLI 組態檔案範例，協助您設定 Greengrass 元件環境。

Hello World (Python)

下列 GDk CLI 組態檔案支援執行 Python 指令碼的 Hello World 元件。此組態檔案使用 zip 建置系統，將元件的 Python 指令碼封裝到 ZIP 檔案中，而 GDk CLI 會上傳為成品。

```
{
  "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)

下列 GDK CLI 組態檔案支援執行 Java 應用程式的 Hello World 元件。此組態檔案使用maven建置系統，將元件的 Java 原始碼封裝到 GDK CLI 上傳為成品的 JAR 檔案。

```
{
  "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"
}
```

社群元件

[Greengrass 軟體目錄](#)中的數個社群元件使用 GDK CLI。您可以在這些元件的儲存庫中探索 GDK CLI 組態檔案。

檢視社群元件的 GDK CLI 組態檔案

1. 執行下列命令，列出使用 GDK CLI 的社群元件。

```
gdk component list --repository
```

回應會列出使用 GDK CLI 的每個社群元件的 GitHub 儲存庫名稱。awslabs 組織中存在每個儲存庫。

```
[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. 在下列 URL 開啟社群元件的 GitHub 儲存庫。將 *community-component-name* 取代為上一個步驟的社群元件名稱。

```
https://github.com/awslabs/community-component-name
```

Greengrass 命令列界面

Greengrass 命令列界面 (CLI) 可讓您與裝置上的 AWS IoT Greengrass Core 互動，在本機開發元件和偵錯問題。例如，您可以使用 Greengrass CLI 來建立本機部署，並重新啟動核心裝置上的元件。

部署 [Greengrass CLI 元件](#) (aws.greengrass.Cli)，在您的核心裝置上安裝 Greengrass CLI。

Important

建議您僅在開發環境中使用此元件，而不是在生產環境中。此元件可讓您存取生產環境中通常不需要的資訊和操作。僅將此元件部署到您需要的核心裝置，以遵循最低權限原則。

主題

- [安裝 Greengrass CLI](#)
- [Greengrass CLI 命令](#)

安裝 Greengrass CLI

您可以使用下列其中一種方式安裝 Greengrass CLI：

- 當您第一次在裝置上設定 AWS IoT Greengrass Core 軟體時，請使用 `--deploy-dev-tools` 引數。您也必須指定 `--provision true` 才能套用此引數。
- 在裝置上部署 Greengrass CLI 元件 (`aws.greengrass.Cli`)。

本節說明部署 Greengrass CLI 元件的步驟。如需有關在初始設定期間安裝 Greengrass CLI 的資訊，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。

先決條件

若要部署 Greengrass CLI 元件，您必須符合下列要求：

- AWS IoT Greengrass 在核心裝置上安裝和設定的核心軟體。如需詳細資訊，請參閱[教學課程：入門 AWS IoT Greengrass V2](#)。
- 若要使用 AWS CLI 部署 Greengrass CLI，您必須已安裝並設定 AWS CLI。如需詳細資訊，請參閱《AWS Command Line Interface 使用者指南》中的[設定 AWS CLI](#)。
- 您必須獲得使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動的授權。執行下列其中一項操作以使用 Greengrass CLI：
 - 使用執行 AWS IoT Greengrass Core 軟體的系統使用者。
 - 使用具有根或管理許可的使用者。在 Linux 核心裝置上，您可以使用 `sudo` 來取得根許可。
 - 當您部署元件時，請使用您在 `AuthorizedPosixGroups` 或 `AuthorizedWindowsGroups` 組態參數中指定的群組中的系統使用者。如需詳細資訊，請參閱 [Greengrass CLI 元件組態](#)。

部署 Greengrass CLI 元件

完成下列步驟，將 Greengrass CLI 元件部署到您的核心裝置：

部署 Greengrass CLI 元件（主控台）

1. 登入 [AWS IoT Greengrass 主控台](#)。
2. 在導覽功能表中，選擇元件。
3. 在元面頁面上的公用元件索引標籤上，選擇 `aws.greengrass.Cli`。
4. 在 `aws.greengrass.Cli` 頁面中，選擇部署。
5. 從新增至部署中，選擇建立新部署。
6. 在指定目標頁面的部署目標下，於目標名稱清單中，選擇您要部署的 Greengrass 群組，然後選擇下一步。

7. 在選取元件頁面上，確認已選取aws.greengrass.Cli元件，然後選擇下一步。
8. 在設定元件頁面上，保留預設組態設定，然後選擇下一步。
9. 在設定進階設定頁面上，保留預設組態設定，然後選擇下一步。
10. 在檢閱頁面上，按一下部署

部署 Greengrass CLI 元件 (AWS CLI)

1. 在您的裝置上，建立 deployment.json 檔案以定義 Greengrass CLI 元件的部署組態。此檔案看起來應該如下所示：

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.16.1",
      "configurationUpdate": {
        "merge": "{ \"AuthorizedPosixGroups\": \"<group1>, <group2>, ..., <groupN>\", \"AuthorizedWindowsGroups\": \"<group1>, <group2>, ..., <groupN>\" }"
      }
    }
  }
}
```

- 在 target 欄位中，*targetArn*以下列格式將物件或物群組的 Amazon Resource Name (ARN) 取代為目標部署：
 - 物件：arn:aws:iot:*region*:*account-id*:thing/*thingName*
 - 物件群組：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 在aws.greengrass.Cli元件物件中，指定值，如下所示：

version

Greengrass CLI 元件的版本。

configurationUpdate.AuthorizedPosixGroups

(選用) 字串，其中包含以逗號分隔的系統群組清單。您授權這些系統群組使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動。您可以指定群組名稱或群組 IDs。例如，會group1,1002,group3授權三個系統群組 (group1、1002和 group3) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，則可以使用 Greengrass CLI 做為根使用者 (sudo) 或執行 AWS IoT Greengrass Core 軟體的系統使用者。

`configurationUpdate.AuthorizedWindowsGroups`

(選用) 字串，其中包含以逗號分隔的系統群組清單。您授權這些系統群組使用 Greengrass CLI 與 AWS IoT Greengrass Core 軟體互動。您可以指定群組名稱或群組 IDs。例如，`group1,1002,group3` 授權三個系統群組 (group1、1002 和 group3) 使用 Greengrass CLI。

如果您未指定任何要授權的群組，則可以使用 Greengrass CLI 做為管理員或執行 AWS IoT Greengrass Core 軟體的系統使用者。

2. 執行下列命令，在裝置上部署 Greengrass CLI 元件：

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/to/deployment.json
```

在安裝期間，元件會在您裝置上的 `greengrass-cli /greengrass/v2/bin` 資料夾中新增符號連結至 `greengrass-cli`，然後您從此路徑執行 Greengrass CLI。若要在沒有絕對路徑的情況下執行 Greengrass CLI，請將 `/greengrass/v2/bin` 資料夾新增至 PATH 變數。若要驗證 Greengrass CLI 安裝，請執行下列命令：

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

您應該會看到下列輸出：

```
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.
```

如果 `greengrass-cli` 找不到，則部署可能已無法安裝 Greengrass CLI。如需詳細資訊，請參閱[故障診斷 AWS IoT Greengrass V2](#)。

Greengrass CLI 命令

Greengrass CLI 提供命令列界面，可在本機與 AWS IoT Greengrass 核心裝置互動。Greengrass CLI 命令使用以下格式。

```
$ greengrass-cli <command> <subcommand> [arguments]
```

根據預設，`/greengrass/v2/bin/` 資料夾中的 `greengrass-cli` 可執行檔會與 `/greengrass/v2` 資料夾中執行 AWS IoT Greengrass 的核心軟體版本互動。如果您呼叫未放置在此位置的可執行檔，或您想要在不同位置與 AWS IoT Greengrass Core 軟體互動，則必須使用下列其中一種方法來明確指定您要互動之 AWS IoT Greengrass Core 軟體的根路徑：

- 將 `GGC_ROOT_PATH` 環境變數設為 `/greengrass/v2`。
- 將 `--ggcRootPath /greengrass/v2` 引數新增至您的命令，如下列範例所示。

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

您可以搭配任何命令使用下列引數：

- 使用 `--help` 以取得特定 Greengrass CLI 命令的相關資訊。
- 使用 `--version` 以取得 Greengrass CLI 版本的相關資訊。

本節說明 Greengrass CLI 命令，並提供這些命令的範例。每個命令的摘要會顯示其引數及其用量。選用引數以方括號顯示。

可用命令

- [元件](#)
- [部署](#)
- [日誌](#)
- [get-debug-password](#)

元件

使用 component 命令與核心裝置上的本機元件互動。

子命令

- [詳細資訊](#)
- [列出](#)
- [重新啟動](#)
- [stop](#)

詳細資訊

擷取一個元件的版本、狀態和組態。

概要

```
greengrass-cli component details --name <component-name>
```

Arguments (引數)

--name、-n。 元件名稱。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli component details --name MyComponent

Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

列出

擷取裝置上安裝的每個元件的名稱、版本、狀態和組態。

概要

```
greengrass-cli component list
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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"}
```

重新啟動

重新啟動元件。

概要

```
greengrass-cli component restart --names <component-name>,...
```

Arguments (引數)

--names、-n。 元件名稱。至少需要一個元件名稱。您可以指定其他元件名稱，以逗號分隔每個名稱。

輸出

無

stop

停止執行元件。

概要

```
greengrass-cli component stop --names <component-name>,...
```

Arguments (引數)

--names、-n。 元件名稱。至少需要一個元件名稱。您可以視需要指定其他元件名稱，以逗號分隔每個名稱。

輸出

無

部署

使用 deployment 命令與核心裝置上的本機元件互動。

若要監控本機部署的進度，請使用 status 子命令。您無法使用 主控台監控本機部署的進度。

子命令

- [建立](#)

- [取消](#)
- [列出](#)
- [status](#)

建立

使用指定的元件配方、成品和執行期引數建立或更新本機部署。

概要

```
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]>]
```

Arguments (引數)

- --recipeDir、-r。 包含元件配方檔案之資料夾的完整路徑。
- --artifactDir、-a。 資料夾的完整路徑，其中包含您要包含在部署中的成品檔案。成品資料夾必須包含下列目錄結構：

```
/path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
```

- --update-config、-c。 部署的組態引數，以 JSON 字串或 JSON 檔案的形式提供。JSON 字串的格式應該如下：

```
{ \
  "componentName": { \
    "MERGE": {"config-key": "config-value"}, \
    "RESET": ["path/to/reset/"] \
  } \
}
```

MERGE 和 區分RESET大小寫，且必須是大寫。

- `--groupId`、`-g`。部署的目標物件群組。
- `--merge`、`-m`。您要新增或更新的目標元件名稱和版本。您必須以格式提供元件資訊 `<component>=<version>`。為每個要指定的額外元件使用單獨的引數。如有需要，請使用 `--runWith` 引數提供執行元件的 `posixGroup`、`posixUser` 和 `windowsUser` 資訊。
- `--runWith`。執行一般或 Lambda 元件的 `posixUser`、`posixGroup`、和 `windowsUser` 資訊。您必須以格式提供此資訊 `<component>: {posixUser|windowsUser}=<user>[:<posixGroup>]`。例如，您可以指定 `HelloWorld:posixUser=ggc_user:ggc_group` 或 `HelloWorld:windowsUser=ggc_user`。為每個要指定的其他選項使用單獨的引數。

如需詳細資訊，請參閱[設定執行元件的使用者](#)。

- `--systemLimits`。系統資源限制，適用於核心裝置上的一般和非容器 Lambda 元件程序。您可以設定每個元件程序可以使用的 CPU 和 RAM 用量上限。指定序列化 JSON 物件或 JSON 檔案的檔案路徑。JSON 物件必須具有下列格式。

```
{ \
  "componentName": { \
    "cpus": cpuTimeLimit, \
    "memory": memoryLimitInKb \
  } \
}
```

您可以為每個元件設定下列系統資源限制：

- `cpus` – 此元件的程序可在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定為 2，以將此元件的程序限制為每個 CPU 核心的 50% 用量。在具有 1 個 CPU 核心的裝置上，您可以將此值設定為 0.25，以將此元件的程序限制為 CPU 的 25% 用量。如果您將此值設定為大於 CPU 核心數量的數字，則 AWS IoT Greengrass Core 軟體不會限制元件的 CPU 用量。
- `memory` – 此元件的程序可在核心裝置上使用的 RAM 數量上限（以 KB 為單位）。

如需詳細資訊，請參閱[設定元件的系統資源限制](#)。

此功能適用於 Linux 核心裝置上的 [Greengrass 核元件](#) 和 Greengrass CLI 的 v2.4.0 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

- `--remove`。您要從本機部署中移除的目標元件名稱。若要移除從雲端部署合併的元件，您必須以下列格式提供目標物件群組的群組 ID：

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`。定義部署失敗時所採取的動作。您可以指定兩個動作：
 - ROLLBACK –
 - DO_NOTHING –

此功能適用於的 v2.11.0 和更新版本。 [Greengrass 核](#)

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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
```

取消

取消指定的部署。

概要

```
greengrass-cli deployment cancel  
  -i <deployment-id>
```

引數

- i。要取消之部署的唯一識別符。部署 ID 會在 create 命令的輸出中傳回。

輸出

- 無

列出

擷取最近 10 個本機部署的狀態。

概要

```
greengrass-cli deployment list
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。根據您的部署狀態，輸出會顯示下列其中一個狀態值：SUCCEEDED、IN_PROGRESS或 FAILED。

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

擷取特定部署的狀態。

概要

```
greengrass-cli deployment status -i <deployment-id>
```

Arguments (引數)

-i。部署的 ID。

輸出

下列範例顯示執行此命令時產生的輸出。根據您的部署狀態，輸出會顯示下列其中一個狀態值：SUCCEEDED、IN_PROGRESS或 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
```

日誌

使用 logs 命令來分析核心裝置上的 Greengrass 日誌。

子命令

- [get](#)
- [list-keywords](#)
- [list-log-files](#)

get

收集、篩選和視覺化 Greengrass 日誌檔案。此命令僅支援 JSON 格式的日誌檔案。您可以在 核組態 中指定[記錄格式](#)。

概要

```
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> ]
```

Arguments (引數)

- `--log-dir`、`-ld`。要檢查日誌檔案的目錄路徑，例如 `/greengrass/v2/logs`。請勿與 `--syslog` 搭配使用。為每個要指定的額外目錄使用單獨的引數。您必須使用 `--log-dir` 或 `--log-file` 中的至少一個。您也可以使用兩個引數。
- `--log-file`、`-lf`。您要使用的日誌目錄路徑。為每個要指定的額外目錄使用單獨的引數。您必須使用 `--log-dir` 或 `--log-file` 中的至少一個。您也可以使用兩個引數。
- `--follow`、`-fol`。在日誌更新發生時顯示日誌更新。Greengrass CLI 會繼續執行並從指定的日誌讀取。如果您指定時段，則 Greengrass CLI 會在所有時段結束後停止監控日誌。
- `--filter`、`-f`。要用作篩選條件的關鍵字、規則運算式或鍵值對。將此值提供為字串、規則表達式或索引鍵/值對。為每個要指定的額外篩選條件使用單獨的引數。

評估時，單一引數中指定的多個篩選條件會由 OR 運算子分隔，而其他引數中指定的篩選條件則會與 AND 運算子合併。例如，如果您的命令包含 `--filter "installed" --filter "name=alpha,name=beta"`，則 Greengrass CLI 將篩選和顯示包含關鍵字 `installed` 和具有 `alpha` 或 `beta` 值的 `name` 金鑰的日誌訊息。

- `--time-window`、`-t`。要顯示日誌資訊的時間範圍。您可以同時使用確切的時間戳記和相對位移。您必須以 `<begin-time>`、`<end-time>` 格式提供此資訊。如果您未指定開始時間或結束時間，則該選項的值會預設為目前的系統日期和時間。為每個要指定的額外時段使用單獨的引數。

Greengrass CLI 支援下列時間戳記格式：

- `yyyy-MM-DD`，例如 `2020-06-30`。當您使用此格式時，時間預設為 `00:00:00`。

`yyyyMMDD`，例如 `20200630`。當您使用此格式時，時間預設為 `00:00:00`。

`HH:mm:ss`，例如 `15:30:45`。當您使用此格式時，日期預設為目前的系統日期。

`HH:mm:ssSSS`，例如 `15:30:45`。當您使用此格式時，日期預設為目前的系統日期。

`YYYY-MM-DD'T'HH:mm:ss'Z'`，例如 `2020-06-30T15:30:45Z`。

`YYYY-MM-DD'T'HH:mm:ss`，例如 `2020-06-30T15:30:45`。

`yyyy-MM-dd'T'HH:mm:ss.SSS`，例如 `2020-06-30T15:30:45.250`。

相對偏移會指定與目前系統時間的時段偏移。Greengrass CLI 支援下列格式的相對偏移：`+|- [<value>h|hr|hours][value m|min|minutes][value s|sec|seconds]`。

例如，下列引數指定目前時間 1 小時到 2 小時 15 分鐘之間的時間範圍--time-window -2h15min, -1hr。

- --verbose。顯示日誌訊息中的所有欄位。請勿與 搭配使用--syslog。
- --no-color、-nc。移除顏色編碼。日誌訊息的預設顏色編碼使用粗體紅色文字。僅支援類似 UNIX 的終端機，因為它使用 ANSI 逸出序列。
- --before、-b。在相符的日誌項目前面顯示的行數。預設值為 0。
- --after、-a。在相符的日誌項目之後顯示的行數。預設值為 0。
- --syslog。使用 RFC3164 定義的 syslog 通訊協定來處理所有日誌檔案。請勿搭配 --log-dir 和 使用 --verbose。syslog 通訊協定使用以下格式："<\$Priority>\$Timestamp \$Host \$Logger (\$Class): \$Message"。如果您未指定日誌檔案，則 Greengrass CLI 會從下列位置讀取日誌訊息：/var/log/messages、/var/log/syslog 或 /var/log/system.log。

AWS IoT Greengrass 目前不支援 Windows 核心裝置上的此功能。

- --max-log-queue-size、-m。要配置到記憶體の日誌項目數目上限。使用此選項可最佳化記憶體用量。預設為 100。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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

顯示建議的關鍵字，您可以用來篩選日誌檔案。

概要

```
greengrass-cli logs list-keywords [arguments]
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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

顯示位於指定目錄中的日誌檔案。

概要

```
greengrass-cli logs list-log-files [arguments]
```

Arguments (引數)

--log-dir、-ld。要檢查日誌檔案的目錄路徑。

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ 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

使用 get-debug-password 命令來為 [本機偵錯主控台元件](#) () 列印隨機產生的密碼aws.greengrass.LocalDebugConsole。密碼會在產生後 8 小時過期。

概要

```
greengrass-cli get-debug-password
```

Arguments (引數)

無

輸出

下列範例顯示執行此命令時產生的輸出。

```
$ sudo greengrass-cli get-debug-password  
  
Username: debug  
Password: bEDp3M0Hdj8ou2w5de_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
```

使用 AWS IoT Greengrass 測試架構

Greengrass 測試架構 (GTF) 是支援從客戶角度 end-to-end 自動化的建置區塊集合。GTF 使用 [Cucumber](#) 做為功能驅動程式。AWS IoT Greengrass 會使用相同的建置區塊來驗證各種裝置上的軟體變更。如需詳細資訊，請參閱 [Github 上的 Greengrass 測試架構](#)。

GTF 使用 Cucumber 實作，這是用來執行自動化測試的工具，可鼓勵元件的行為驅動開發 (BDD)。在 Cucumber 中，此系統的功能概述於稱為的特殊檔案類型中 feature。每個功能都以稱為案例的人類可讀格式描述，這些案例是可轉換為自動化測試的規格。每個案例都概述為一系列步驟，使用稱為 Gherkin 的網域特定語言來定義此系統的互動和結果。[Gherkin 步驟](#) 會使用稱為步驟定義的方法連結至程式設計程式碼，該方法將規格硬式路由至測試流程。GTF 中的步驟定義是使用 Java 實作。

主題

- [運作方式](#)
- [變更記錄](#)
- [Greengrass 測試架構組態選項](#)
- [教學課程：使用 Greengrass 測試架構和 Greengrass 開發套件執行 end-to-end 測試](#)
- [教學課程：使用可信度測試套件中的可信度測試](#)

運作方式

AWS IoT Greengrass 會將 GTF 分發為由數個 Java 模組組成的獨立 JAR。若要使用 GTF end-to-end 測試，您必須在 Java 專案中實作測試。在 Java 專案中新增測試可站立的 JAR 做為相依性，可讓您使用 GTF 的現有功能，並透過撰寫自己的自訂測試案例來擴展它。若要執行自訂測試案例，您可以建置 Java 專案，並使用中所述的組態選項執行目標 JAR [Greengrass 測試架構組態選項](#)。

GTF 獨立 JAR

Greengrass 使用 Cloudfront 做為 [Maven](#) 儲存庫來託管不同版本的 GTF 獨立 JAR。如需 GTF 版本的完整清單，請參閱 [GTF 版本](#)。

GTF 獨立 JAR 包含下列模組。這不限於這些模組。您可以分別在專案中挑選和選擇每個相依性，或使用 [測試獨立 JAR 檔案](#) 同時包含所有相依性。

- `aws-greengrass-testing-resources`：本單元提供在測試過程中管理 AWS 資源生命週期的抽象概念。您可以使用此參數來定義自訂 AWS 資源，以便 GTF ResourceSpec 為您建立和移除這些資源。
- `aws-greengrass-testing-platform`：此模組會在測試生命週期期間為待測裝置提供平台層級抽象。它包含 API，APIs 用於與獨立於平台的作業系統互動，並可用於模擬在裝置 shell 中執行的命令。
- `aws-greengrass-testing-components`：此模組包含用於測試 Greengrass 核心功能的範例元件，例如部署、IPC 和其他功能。
- `aws-greengrass-testing-features`：此模組包含可重複使用的常見步驟及其定義，用於在 Greengrass 環境中測試內的。

主題

- [變更記錄](#)
- [Greengrass 測試架構組態選項](#)
- [教學課程：使用 Greengrass 測試架構和 Greengrass 開發套件執行 end-to-end 測試](#)
- [教學課程：使用可信度測試套件中的可信度測試](#)

變更記錄

下表說明 GTF 每個版本中的變更。如需詳細資訊，請參閱 GitHub 上的 [GTF 版本頁面](#)。

版本	變更
1.2.0	新功能 • 新增網路相關步驟，以在測試期間設定 MQTT 和網際網路網路連線。 • 新增系統指標步驟以監控裝置 RAM 和 CPU 使用。 錯誤修正和改善 • Greengrass CLI 本機部署步驟會重試，直到成功為止。 • 測試會正常停止 Greengrass 核，而不是將其刪除。 • 新增改善，其中 GTF 輪詢 AWS IoT 登入資料端點，直到可擷取物件和角色別名的登入資料為止。 • 修正缺少的成品和配方目錄。此版本也會修正缺少的元件版本。

版本	變更
	- 修正當 Docker 映像不存在時，在 Docker 映像清除期間 GTF 失敗的問題。 - 新增 CURRENT 關鍵字做為元件的版本。
1.1.0	新功能 - 新增使用組態安裝自訂元件的功能。這需要自訂元件的配方。 - 新增使用自訂組態更新本機部署的功能。 錯誤修正和改善 - 修正日誌內容 GTF 版本不一致的問題。
1.0.0	初始版本。

Greengrass 測試架構組態選項

GTF 組態選項

Greengrass 測試架構 (GTF) 可讓您在 end-to-end 測試程序啟動期間設定特定參數，以協調測試流程。您可以將這些組態選項指定為 GTF 獨立 JAR 的 CLI 引數。

GTF 1.1.0 版及更新版本提供下列組態選項。

- `additional-plugins` – (選用) 其他小黃瓜外掛程式
- `aws-region` – 以服務的特定區域端點為 AWS 目標。預設為 AWS SDK 探索的內容。
- `credentials-path` – 選用 AWS 的設定檔登入資料路徑。預設為主機環境上探索的登入資料。
- `credentials-path-rotation` – 登入資料的 AWS 選用輪換持續時間。預設為 15 分鐘或 PT15M。
- `csr-path` – 產生裝置憑證的 CSR 路徑。
- `device-mode` – 正在測試的目標裝置。預設為本機裝置。
- `env-stage` – 以 Greengrass 的部署環境為目標。預設為生產。
- `existing-device-cert-arn` – 您要用來做為 Greengrass 裝置憑證的現有憑證。
- `feature-path` – 包含其他特徵檔案的檔案或目錄。預設為不使用其他功能檔案。
- `gg-cli-version` – 覆寫 Greengrass CLI 的版本。預設為中找到的值 `ggc.version`。

- `gg-component-bucket` – 存放 Greengrass 元件的現有 Amazon S3 儲存貯體的名稱。
- `gg-component-overrides` – Greengrass 元件覆寫的清單。
- `gg-persist` – 測試執行後要保留的測試元素清單。預設行為是不保留任何內容。接受的值為：`aws.resources`、`installed.software`和`generated.files`。
- `gg-runtime` – 影響測試如何與測試資源互動的值清單。這些值會取代 `gg.persist` 參數。如果預設值為空，則假設所有測試資源都由測試案例管理，包括安裝的 Greengrass 執行期。接受的值為：`aws.resources`、`installed.software`和`generated.files`。
- `ggc-archive` – 封存 Greengrass 核元件的路徑。
- `ggc-install-root` – 安裝 Greengrass 核元件的目錄。預設為 `test.temp.path` 和 `test run` 資料夾。
- `ggc-log-level` – 設定測試執行的 Greengrass 核日誌層級。預設為「INFO」。
- `ggc-tes-rolename` – Core AWS IoT Greengrass 將擔任以存取 AWS 服務的 IAM 角色。如果指定名稱的角色不存在，則會建立一個角色和預設存取政策。
- `ggc-trusted-plugins` – 需要新增至 Greengrass 的信任外掛程式路徑（主機上）的逗號分隔清單。若要提供 DUT 本身的路徑，請在路徑前面加上 `'dut:'`。
- `ggc-user-name` – Greengrass 核的 `user: group posixUser` 值。預設為目前登入的使用者名稱。
- `ggc-version` – 覆寫執行中 Greengrass 核元件的版本。預設為 `ggc.archive` 中的值。
- `log-level` – 測試執行的日誌層級。預設為「INFO」。
- `parallel-config` – 批次索引和批次數量的集合，做為 JSON 字串。批次索引的預設值為 0，批次數量為 1。
- `proxy-url` – 設定所有測試以透過此 URL 路由流量。
- `tags` – 僅執行功能標籤。可與 `'&'` 交集
- `test-id-prefix` – 套用至所有測試特定資源的通用字首，包括 AWS 資源名稱和標籤。預設為「gg」字首。
- `test-log-path` – 將包含整個測試執行結果的目錄。預設為「testResults」。
- `test-results-json` – 標記以判斷是否產生寫入磁碟的 Cucumber JSON 報告。預設為 `true`。
- `test-results-log` – 用來判斷主控台輸出是否寫入磁碟的旗標。預設為 `false`。
- `test-results-xml` – 標記以判斷是否產生寫入磁碟的 JUnit XML 報告。預設為 `true`。
- `test-temp-path` – 產生本機測試成品的目錄。預設為字首為 `gg` 測試的隨機暫存目錄。
- `timeout-multiplier` – 提供給所有測試逾時的乘數。預設值為 1.0。

教學課程：使用 Greengrass 測試架構和 Greengrass 開發套件執行end-to-end測試

AWS IoT Greengrass 測試架構 (GTF) 和 Greengrass 開發套件 (GDK) 為開發人員提供執行end-to-end測試的方法。您可以完成此教學課程，以使用元件初始化 GDK 專案、使用end-to-end測試模組初始化 GDK 專案，以及建置自訂測試案例。建置自訂測試案例之後，您就可以執行測試。

在此教學課程中，您將執行下列操作：

1. 使用元件初始化 GDK 專案。
2. 使用end-to-end測試模組初始化 GDK 專案。
3. 建置自訂測試案例。
4. 將標籤新增至新的測試案例。
5. 建置測試 JAR。
6. 執行 測試。

主題

- [先決條件](#)
- [步驟 1：使用元件初始化 GDK 專案](#)
- [步驟 2：使用end-to-end測試模組初始化 GDK 專案](#)
- [步驟 3：建置自訂測試案例](#)
- [步驟 4：將標籤新增至新的測試案例](#)
- [步驟 5：建置測試 JAR](#)
- [步驟 6：執行測試](#)
- [範例：建置自訂測試案例](#)

先決條件

為了完成本教學，您需要以下項目：

- GDK 1.3.0 版或更新版本
- Java
- Maven
- Git

步驟 1：使用元件初始化 GDK 專案

- 使用 GDK 專案初始化空資料夾。執行下列命令，下載在 Python 中實作的 HelloWorld 元件。

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

此命令 HelloWorld 會在目前目錄中建立名為 的新目錄。

步驟 2：使用 end-to-end 測試模組初始化 GDK 專案

- GDK 可讓您下載由功能和步驟實作組成的測試模組範本。執行下列命令以開啟 HelloWorld 目錄，並使用測試模組初始化現有的 GDK 專案。

```
cd HelloWorld
gdk test-e2e init
```

此命令會在 目錄中建立新的名為 gg-e2e-tests 的 HelloWorld 目錄。此測試目錄是 [Maven](#) 專案，其相依於 Greengrass 測試獨立 JAR。

步驟 3：建置自訂測試案例

撰寫自訂測試案例廣泛包含兩個步驟：使用測試案例建立特徵檔案並實作步驟定義。如需建置自訂測試案例的範例，請參閱 [範例：建置自訂測試案例](#)。使用下列步驟來建置您的自訂測試案例：

1. 使用測試案例建立特徵檔案

功能通常會描述正在測試之軟體的特定功能。在 Cucumber 中，每個功能都會指定為個別功能檔案，其中包含標題、詳細說明，以及一或多個稱為案例的特定案例範例。每個案例都包含標題、詳細說明，以及定義互動和預期結果的一系列步驟。使用 "given"、"when" 和 "then" 關鍵字以結構化格式撰寫案例。

2. 實作步驟定義

步驟定義以純語言將 [Gherkin 步驟](#) 連結至程式設計程式碼。當 Cucumber 在案例中識別 Gherkin 步驟時，它會尋找要執行的相符步驟定義。

步驟 4：將標籤新增至新的測試案例

- 您可以將標籤指派給特徵和案例，以組織測試程序。您可以使用標籤來分類案例子集，也可以依條件選取要執行的勾點。功能和案例可以有多个以空格分隔的標籤。

在此範例中，我們使用 HelloWorld 元件。

在特徵檔案中，在標籤@HelloWorld旁新增名為 的新@Sample標籤。

```
@Sample @HelloWorld
Scenario: As a developer, I can create a component and deploy it on my device
....
```

步驟 5：建置測試 JAR

- 建置 元件。您必須先建置元件，才能建置測試模組。

```
gdk component build
```

- 使用以下命令建置測試模組。此命令將在 greengrass-build 資料夾中建置測試 JAR。

```
gdk test-e2e build
```

步驟 6：執行測試

當您執行自訂測試案例時，GTF 會自動執行測試的生命週期，並管理測試期間建立的資源。它會先佈建待測裝置 (DUT) 做為 AWS IoT 物件，並在其上安裝 Greengrass 核心軟體。然後，它會 HelloWorld 使用該路徑中指定的配方來建立新的名為 的元件。然後，HelloWorld 元件會透過 Greengrass 物件部署部署到核心裝置上。如果部署成功，則會對其進行驗證。如果部署成功，部署狀態會在 COMPLETED 3 分鐘內變更為 。

- 前往專案目錄中的 gdk-config.json 檔案，以 HelloWorld 標籤鎖定測試目標。使用下列命令更新 test-e2e 金鑰。

```
"test-e2e":{
  "gtf_options" : {
    "tags":"HelloWorld"
  }
}
```

```
}
```

2. 執行測試之前，您必須提供 AWS 登入資料給主機裝置。GTF 會在測試過程中使用這些登入資料來管理 AWS 資源。請確定您提供的角色具有許可，可自動化測試中包含的必要操作。

執行下列命令以提供 AWS 登入資料。

- 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. 使用以下命令執行測試。

```
gdk test-e2e run
```

此命令會在 greengrass-build 資料夾中下載最新版本的 Greengrass 核，並使用它執行測試。此命令也會僅以具有 HelloWorld 標籤的案例為目標，並產生這些案例的報告。您會在測試結束時看到在此測試期間建立 AWS 的資源遭到捨棄。

範例：建置自訂測試案例

Example

GDK 專案中下載的測試模組包含範例功能和步驟實作檔案。

在下列範例中，我們會建立功能檔案，以測試 Greengrass 軟體的物件部署功能。我們會使用透過 Greengrass 部署元件的案例來部分測試此功能的功能 AWS 雲端。這是一系列步驟，可協助我們了解此使用案例的互動和預期結果。

1. 建立特徵檔案

導覽至目前目錄中的 `gg-e2e-tests/src/main/resources/greengrass/features` 資料夾。您可以找到 `component.feature` 類似下列範例的範例。

在此功能檔案中，您可以測試 Greengrass 軟體的物件部署功能。您可以使用透過 Greengrass 雲端執行元件部署的案例來部分測試此功能的功能。該案例是一系列步驟，有助於了解此使用案例的互動和預期結果。

```
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
```

GTF 包含所有下列步驟的步驟定義，但名為 `And I call my custom step` 的步驟除外：

2. 實作步驟定義

GTF 獨立 JAR 包含除了一個步驟以外的所有步驟的步驟定義：`And I call my custom step`。您可以在測試模組中實作此步驟。

導覽至測試檔案的原始程式碼。您可以使用下列命令，使用步驟定義來連結自訂步驟。

```
@And("I call my custom step")  
public void customStep() {  
    System.out.println("My custom step was called ");  
}
```

教學課程：使用可信度測試套件中的可信度測試

AWS IoT Greengrass 測試架構 (GTF) 和 Greengrass 開發套件 (GDK) 為開發人員提供執行end-to-end測試的方法。您可以完成此教學課程，以使用元件初始化 GDK 專案、使用end-to-end測試模組初始化 GDK 專案，以及使用可信度測試套件中的可信度測試。建置自訂測試案例之後，您就可以執行測試。

可信度測試是由 Greengrass 提供的一般測試，可驗證基本元件行為。這些測試可以修改或擴展，以滿足更具體的元件需求。

在本教學課程中，我們將使用 HelloWorld 元件。如果您使用的是其他元件，請將 HelloWorld 元件取代為您的元件。

在此教學課程中，您將執行下列操作：

1. 使用元件初始化 GDK 專案。
2. 使用end-to-end測試模組初始化 GDK 專案。
3. 使用可信度測試套件中的測試。
4. 將標籤新增至新的測試案例。
5. 建置測試 JAR。
6. 執行 測試。

主題

- [先決條件](#)
- [步驟 1：使用元件初始化 GDK 專案](#)
- [步驟 2：使用end-to-end測試模組初始化 GDK 專案](#)
- [步驟 3：使用可信度測試套件中的測試](#)
- [步驟 4：將標籤新增至新的測試案例](#)
- [步驟 5：建置測試 JAR](#)
- [步驟 6：執行測試](#)
- [範例：使用可信度測試](#)

先決條件

為了完成本教學，您需要以下項目：

- GDK 1.6.0 版或更新版本

- Java
- Maven
- Git

步驟 1：使用元件初始化 GDK 專案

- 使用 GDK 專案初始化空資料夾。執行下列命令，下載在 Python 中實作的 HelloWorld 元件。

```
gdk component init -t HelloWorld -l python -n HelloWorld
```

此命令 HelloWorld 會在目前目錄中建立名為 的新目錄。

步驟 2：使用 end-to-end 測試模組初始化 GDK 專案

- GDK 可讓您下載由功能和步驟實作組成的測試模組範本。執行下列命令以開啟 HelloWorld 目錄，並使用測試模組初始化現有的 GDK 專案。

```
cd HelloWorld
gdk test-e2e init
```

此命令會在 目錄中建立新的名為 gg-e2e-tests 的 HelloWorld 目錄。此測試目錄是 [Maven](#) 專案，其相依於 Greengrass 測試獨立 JAR。

步驟 3：使用可信度測試套件中的測試

撰寫可信度測試案例包含使用提供的特徵檔案，並視需要修改案例。如需使用可信度測試的範例，請參閱 [範例：建置自訂測試案例](#)。使用下列步驟來使用可信度測試：

- 使用提供的功能檔案。

導覽至目前目錄中的 gg-e2e-tests/src/main/resources/greengrass/features 資料夾。開啟範例 confidenceTest.feature 檔案以使用可信度測試。

步驟 4：將標籤新增至新的測試案例

- 您可以將標籤指派給功能和案例，以組織測試程序。您可以使用標籤來分類案例子集，也可以依條件選取要執行的勾點。功能和案例可以有多個以空格分隔的標籤。

在此範例中，我們使用 HelloWorld 元件。

每個案例都會以 標記@ConfidenceTest。如果您想要僅執行測試套件的一部分，請變更或新增標籤。每個測試案例都描述在每個可信度測試的頂端。該案例是一系列步驟，有助於了解每個測試案例的互動和預期結果。您可以透過新增自己的步驟或修改現有的步驟來擴展這些測試。

```
@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
         is working as expected
.....
```

步驟 5：建置測試 JAR

1. 建置 元件。您必須先建置元件，才能建置測試模組。

```
gdk component build
```

2. 使用以下命令建置測試模組。此命令將在 greengrass-build 資料夾中建置測試 JAR。

```
gdk test-e2e build
```

步驟 6：執行測試

當您執行可信度測試時，GTF 會自動執行測試的生命週期，並管理測試期間建立的資源。它會先佈建待測裝置 (DUT) 做為 AWS IoT 物件，並在其上安裝 Greengrass 核心軟體。然後，它會HelloWorld使用該路徑中指定的配方來建立新的名為 的元件。然後，HelloWorld元件會透過 Greengrass 物件部署部署到核心裝置上。如果部署成功，則會對其進行驗證。如果部署成功，部署狀態會在 COMPLETED 3 分鐘內變更為 。

1. 前往專案目錄中gdk-config.json的檔案，以 ConfidenceTest標籤或步驟 4 中指定的任何標籤 yo8u 為目標測試。使用下列命令更新test-e2e金鑰。

```
"test-e2e":{
  "gtf_options" : {
    "tags":"ConfidenceTest"
  }
}
```

2. 執行測試之前，您必須提供 AWS 登入資料給主機裝置。GTF 會在測試過程中使用這些登入資料來管理 AWS 資源。請確定您提供的角色具有將測試中包含的必要操作自動化的許可。

執行下列命令以提供 AWS 登入資料。

- 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. 使用以下命令執行測試。

```
gdk test-e2e run
```

此命令會在 greengrass-build 資料夾中下載最新版本的 Greengrass 核，並使用它執行測試。此命令也會僅以具有 ConfidenceTest 標籤的案例為目標，並產生這些案例的報告。您會在測試結束時看到在此測試期間建立 AWS 的資源遭到捨棄。

範例：使用可信度測試

Example

GDK 專案中下載的測試模組包含提供的功能檔案。

在下列範例中，我們使用功能檔案來測試 Greengrass 軟體的物件部署功能。我們透過 Greengrass 執行元件部署的案例，部分測試此功能的功能 AWS 雲端。這是一系列步驟，可協助我們了解此使用案例的互動和預期結果。

- 使用提供的功能檔案。

導覽至目前目錄中的 `gg-e2e-tests/src/main/resources/greengrass/features` 資料夾。您可以找到 `confidenceTest.feature` 類似下列範例的範例。

```
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
```

每個測試案例都描述在每個可信度測試的頂端。該案例是一系列步驟，有助於了解每個測試案例的互動和預期結果。您可以透過新增自己的步驟或修改現有的步驟來擴展這些測試。每個案例都包含可協助您進行這些調整的註解。

開發 AWS IoT Greengrass 元件

您可以在 Greengrass 核心裝置上開發和測試元件。因此，您可以建立和重複您的 AWS IoT Greengrass 軟體，而無需與互動 AWS 雲端。當您完成元件的版本時，您可以在雲端 AWS IoT Greengrass 中將其上傳至 [AWS IoT Greengrass 元件部署至裝置](#)，以便您和您的團隊可以將元件部署到機群中的其他裝置。如需如何部署元件的詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

每個元件都由配方和成品組成。

- 配方

每個元件都包含配方檔案，定義其中繼資料。配方也會指定元件的組態參數、元件相依性、生命週期和平台相容性。元件生命週期會定義安裝、執行和關閉元件的命令。如需詳細資訊，請參閱[AWS IoT Greengrass 元件配方參考](#)。

您可以定義 [JSON](#) 或 [YAML](#) 格式的配方。

- 成品

元件可以有任意數量的成品，這些成品是元件二進位檔。成品可以包含指令碼、編譯程式碼、靜態資源，以及元件使用的任何其他檔案。元件也可以使用元件相依性的成品。

AWS IoT Greengrass 提供預先建置的元件，您可以在應用程式中使用，並部署到裝置。例如，您可以使用串流管理員元件將資料上傳至各種 AWS 服務，或使用 CloudWatch 指標元件將自訂指標發佈至 Amazon CloudWatch。如需詳細資訊，請參閱[AWS提供的元件](#)。

AWS IoT Greengrass 策劃 Greengrass 元件的索引，稱為 Greengrass 軟體目錄。此目錄會追蹤由 Greengrass 社群開發的 Greengrass 元件。您可以從此目錄中下載、修改和部署元件，以建立 Greengrass 應用程式。如需詳細資訊，請參閱[社群元件](#)。

AWS IoT Greengrass 核心軟體會以您在核心裝置上設定的系統使用者和群組身分執行元件 `ggc_group`，例如 `ggc_user` 和 `ggc_group`。這表示元件具有該系統使用者的許可。如果您使用沒有主目錄的系統使用者，則元件無法使用使用主目錄的執行命令或程式碼。這表示您無法使用 `pip install some-library --user` 命令來安裝 Python 套件。如果您遵循[入門教學](#)來設定核心裝置，則您的系統使用者沒有主目錄。如需如何設定執行元件之使用者和群組的詳細資訊，請參閱[設定執行元件的使用者](#)。

Note

AWS IoT Greengrass 使用元件的語意版本。語意版本遵循 `major.minor.patch` 號碼系統。例如，版本 `1.0.0` 代表元件的第一個主要版本。如需詳細資訊，請參閱[語意版本規格](#)。

主題

- [元件生命週期](#)
- [元件類型](#)
- [建立 AWS IoT Greengrass 元件](#)
- [使用本機部署測試 AWS IoT Greengrass 元件](#)

- [發佈元件以部署到您的核心裝置](#)
- [與 AWS 服務互動](#)
- [執行 Docker 容器](#)
- [AWS IoT Greengrass 元件配方參考](#)
- [元件環境變數參考](#)

元件生命週期

元件生命週期會定義 AWS IoT Greengrass Core 軟體用來安裝和執行元件的階段。每個階段都會定義指令碼和其他資訊，以指定元件的行為方式。例如，當您安裝元件時，AWS IoT Greengrass Core 軟體會執行該元件的 `install` 生命週期指令碼。核心裝置上的元件具有下列生命週期狀態：

- NEW – 元件的配方和成品會載入核心裝置上，但元件並未安裝。元件進入此狀態後，會執行其[安裝指令碼](#)。
- INSTALLED – 元件安裝在核心裝置上。元件在執行其[安裝指令碼](#)後會進入此狀態。
- STARTING – 元件正在核心裝置上啟動。元件會在執行其[啟動指令碼](#)時進入此狀態。如果啟動成功，元件會進入 RUNNING 狀態。
- RUNNING – 元件正在核心裝置上執行。元件在執行其[執行指令碼](#)時，或從其啟動指令碼中具有作用中背景程序時，會進入此狀態。
- FINISHED – 元件已成功執行並完成其執行。
- STOPPING – 元件正在停止。元件在執行其[關機指令碼](#)時進入此狀態。
- ERRORED – 元件發生錯誤。當元件進入此狀態時，它會執行[復原指令碼](#)。然後，元件會重新啟動，嘗試恢復正常使用。如果元件在未成功執行的情況下進入 ERRORED 狀態三次，則元件會變成 BROKEN。
- BROKEN – 元件多次發生錯誤，無法復原。您必須再次部署元件才能修正。

元件類型

元件類型指定 AWS IoT Greengrass Core 軟體如何執行元件。元件可以有下列類型：

- Nucleus (`aws.greengrass.nucleus`)

Greengrass 核是提供 AWS IoT Greengrass Core 軟體最低功能的元件。如需詳細資訊，請參閱[Greengrass 核](#)。

- 外掛程式 (`aws.greengrass.plugin`)

Greengrass 核會在與核相同的 Java 虛擬機器 (JVM) 中執行外掛程式元件。當您變更核心裝置上的外掛程式元件版本時，核會重新啟動。若要安裝和執行外掛程式元件，您必須設定 Greengrass 核以系統服務的形式執行。如需詳細資訊，請參閱[將 Greengrass 核設定為系統服務](#)。

提供的數個元件 AWS 是外掛程式元件，可讓它們直接與 Greengrass 核連接。外掛程式元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

- 一般 (`aws.greengrass.generic`)

如果元件定義生命週期，Greengrass 核會執行一般元件的生命週期指令碼。

此類型是自訂元件的預設類型。

- Lambda (`aws.greengrass.lambda`)

Greengrass 核會使用 Lambda [啟動器元件執行 Lambda 函數元件](#)。

當您從 Lambda 函數建立元件時，該元件具有此類型。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

 Note

我們不建議您在 recipe 中指定元件類型。當您建立元件時，會為您 AWS IoT Greengrass 設定類型。

建立 AWS IoT Greengrass 元件

您可以在本機開發電腦或 Greengrass 核心裝置上開發自訂 AWS IoT Greengrass 元件。AWS IoT Greengrass 提供[AWS IoT Greengrass 開發套件命令列介面 \(GDK CLI\)](#)，協助您從預先定義的元件範本和[社群元件](#)建立、建置和發佈元件。您也可以執行內建 shell 命令來建立、建置和發佈元件。從下列選項中選擇，以建立自訂 Greengrass 元件：

- 使用 Greengrass 開發套件 CLI

使用 GDK CLI 在本機開發電腦上開發元件。GDK CLI 會將元件原始程式碼建置並封裝到配方和成品中，您可以將其做為私有元件發佈至 AWS IoT Greengrass 服務。您可以設定 GDK CLI，以便在發佈元件時自動更新元件的版本和成品 URIs，因此您不需要每次更新配方。若要使用 GDK CLI 開發元件，您可以從範本或 [Greengrass 軟體目錄](#)中的社群元件開始。如需詳細資訊，請參閱[AWS IoT Greengrass 開發套件命令列介面](#)。

- 執行內建 shell 命令

您可以執行內建 shell 命令，在本機開發電腦或 Greengrass 核心裝置上開發元件。您可以使用 shell 命令將元件原始程式碼複製或建置為成品。每次建立新版本的元件時，您必須使用新的元件版本建立或更新配方。當您將元件發佈至服務時 AWS IoT Greengrass，您必須將 URI 更新為配方中的每個元件成品。

主題

- [建立元件 \(GDK CLI\)](#)
- [建立元件 \(shell 命令\)](#)

建立元件 (GDK CLI)

遵循本節中的指示，使用 GDK CLI 建立和建置元件。

開發 Greengrass 元件 (GDK CLI)

1. 如果您尚未安裝，請在開發電腦上安裝 GDK CLI。如需詳細資訊，請參閱[安裝或更新 AWS IoT Greengrass 開發套件命令列界面](#)。
2. 將變更為您要建立元件資料夾的資料夾。

Linux or Unix

```
mkdir ~/greengrassv2  
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2  
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2  
cd ~/greengrassv2
```

3. 選擇要下載的元件範本或社群元件。GDK CLI 會下載範本或社群元件，因此您可以從功能範例開始。使用[元件清單](#)命令來擷取可用範本或社群元件的清單。

- 若要列出元件範本，請執行下列命令。回應中的每一行都包含範本的名稱和程式設計語言。

```
gdk component list --template
```

- 若要列出社群元件，請執行下列命令。

```
gdk component list --repository
```

- 建立並變更為 GDK CLI 下載範本或社群元件的元件資料夾。將 *HelloWorld* 取代為元件名稱，或另一個可協助您識別此元件資料夾的名稱。

Linux or Unix

```
mkdir HelloWorld  
cd HelloWorld
```

Windows Command Prompt (CMD)

```
mkdir HelloWorld  
cd HelloWorld
```

PowerShell

```
mkdir HelloWorld  
cd HelloWorld
```

- 將範本或社群元件下載至目前的資料夾。使用 [元件 init](#) 命令。
 - 若要從範本建立元件資料夾，請執行下列命令。將 *HelloWorld* 取代為範本的名稱，並將 *python* 取代為程式設計語言的名稱。

```
gdk component init --template HelloWorld --language python
```

- 若要從社群元件建立元件資料夾，請執行下列命令。將 *ComponentName* 取代為社群元件的名稱。

```
gdk component init --repository ComponentName
```

 Note

如果您使用 GDK CLI 1.0.0 版，您必須在空的資料夾中執行此命令。GDK CLI 會將範本或社群元件下載到目前的資料夾。

如果您使用 GDK CLI 1.1.0 版或更新版本，您可以指定 `--name` 引數來指定 GDK CLI 下載範本或社群元件的資料夾。如果您使用此引數，請指定不存在的資料夾。GDK CLI 會為您建立資料夾。如果您未指定此引數，GDK CLI 會使用目前的資料夾，該資料夾必須是空的。

6. GDK CLI 會從名為的 [GDK CLI 組態檔案](#) 讀取 `gdk-config.json`，以建置和發佈元件。此組態檔案存在於元件資料夾的根目錄中。上一個步驟會為您建立此檔案。在此步驟中，您會 `gdk-config.json` 使用元件的相關資訊進行更新。請執行下列操作：

- a. 在文字編輯器中開啟 `gdk-config.json`。
- b. (選用) 變更元件的名稱。元件名稱是 `component` 物件中的金鑰。
- c. 變更元件的作者。
- d. (選用) 變更元件的版本。請指定下列其中一項：
 - `NEXT_PATCH` – 當您選擇此選項時，GDK CLI 會在您發佈元件時設定版本。GDK CLI 會查詢 AWS IoT Greengrass 服務，以識別元件的最新發佈版本。然後，它會將版本設定為該版本之後的下一個修補程式版本。如果您之前尚未發佈元件，GDK CLI 會使用版本 `1.0.0`。

如果您選擇此選項，則無法使用 [Greengrass CLI](#) 在本機部署和測試元件到執行 AWS IoT Greengrass 核心軟體的本機開發電腦。若要啟用本機部署，您必須改為指定語意版本。

 - 語意版本，例如 `1.0.0`。語意版本使用 `major.minor.patch` 編號系統。如需詳細資訊，請參閱 [語意版本規格](#)。

如果您在想要部署和測試元件的 Greengrass 核心裝置上開發元件，請選擇此選項。您必須使用特定版本建置元件，才能使用 [Greengrass CLI](#) 建立本機部署。
- e. (選用) 變更元件的建置組態。建置組態定義 GDK CLI 如何將元件的來源建置為成品。請從下列選項中選擇 `build_system`：
 - `zip` – 將元件的資料夾封裝成 ZIP 檔案，以定義為元件的唯一成品。針對下列元件類型選擇此選項：
 - 使用解譯程式設計語言的元件，例如 Python 或 JavaScript。

- 封裝程式碼以外檔案的元件，例如機器學習模型或其他資源。

GDK CLI 會將元件的資料夾壓縮成與元件資料夾同名的 zip 檔案。例如，如果元件資料夾的名稱為 HelloWorld，則 GDK CLI 會建立名為的 zip 檔案HelloWorld.zip。

 Note

如果您在 Windows 裝置上使用 GDK CLI 1.0.0 版，元件資料夾和 zip 檔案名稱只能包含小寫字母。

當 GDK CLI 將元件的資料夾壓縮為 zip 檔案時，它會略過下列檔案：

- gdk-config.json 檔案
- 配方檔案 (recipe.json 或 recipe.yaml)
- 建置資料夾，例如 greengrass-build
- maven – 執行 `mvn clean package` 命令，將元件的來源建置為成品。針對使用 [Maven](#) 的元件選擇此選項，例如 Java 元件。

在 Windows 裝置上，此功能適用於 GDK CLI 1.1.0 版及更新版本。

- gradle – 執行 `gradle build` 命令，將元件的來源建置為成品。針對使用 [Gradle](#) 的元件選擇此選項。此功能適用於 GDK CLI 1.1.0 版及更新版本。

gradle 建置系統支援 Kotlin DSL 做為建置檔案。此功能適用於 GDK CLI 1.2.0 版及更新版本。

- gradlew – 執行 `gradlew` 命令，將元件的來源建置為成品。針對使用 [Gradle Wrapper](#) 的元件選擇此選項。

此功能適用於 GDK CLI 1.2.0 版及更新版本。

- custom – 執行自訂命令，將元件的來源建置到配方和成品中。在 `custom_build_command` 參數中指定自訂命令。
- f. 如果您 custom 為指定 `build_system`，請將 `custom_build_command` 新增至 build 物件。在中 `custom_build_command`，指定單一字串或字串清單，其中每個字串都是命令中的字詞。例如，若要執行 C++ 元件的自訂建置命令，您可以指定 `["cmake", "--build", "build", "--config", "Release"]`。
- g. 如果您使用 GDK CLI 1.1.0 版或更新版本，您可以指定 `--bucket` 引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳至名為的

S3 儲存貯體 *bucket-region-accountId*，其中 *###* 體和 *##* 是您在 中指定的值 *gdk-config.json*，而 *accountId* 是您的 AWS 帳戶 ID。如果儲存貯體不存在，GDK CLI 會建立儲存貯體。

變更元件的發佈組態。請執行下列操作：

- i. 指定用於託管元件成品的 S3 儲存貯體名稱。
- ii. 指定 AWS 區域 GDK CLI 發佈元件的。

當您完成此步驟時，*gdk-config.json* 檔案看起來可能會類似下列範例。

```
{
  "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. 更新名為 *recipe.yaml* 或 的元件配方檔案 *recipe.json*。請執行下列操作：

- a. 如果您下載了使用 *zip* 建置系統的範本或社群元件，請檢查 *zip* 成品名稱是否符合元件資料夾的名稱。GDK CLI 會將元件資料夾壓縮為與元件資料夾同名的 *zip* 檔案。配方包含元件成品清單中的 *zip* 成品名稱，以及使用 *zip* 成品中檔案的生命週期指令碼。更新 *Artifacts* 和 *Lifecycle* 定義，讓 *zip* 檔案名稱符合元件資料夾的名稱。下列部分配方範例會反白顯示 *Artifacts* 和 *Lifecycle* 定義中的 *zip* 檔案名稱。

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. (選用) 更新元件描述、預設組態、成品、生命週期指令碼和平台支援。如需詳細資訊，請參閱[AWS IoT Greengrass 元件配方參考](#)。

完成此步驟後，配方檔案看起來可能會類似下列範例。

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. 開發和建置 Greengrass 元件。[元件建置](#)命令會在元件greengrass-build資料夾中產生配方和成品。執行下列命令。

```
gdk component build
```

當您準備好測試元件時，請使用 GDK CLI 將其發佈至服務 AWS IoT Greengrass。然後，您可以將元件部署到 Greengrass 核心裝置。如需詳細資訊，請參閱[發佈元件以部署到您的核心裝置](#)。

建立元件 (shell 命令)

遵循本節中的指示，建立配方和成品資料夾，其中包含多個元件的原始程式碼和成品。

開發 Greengrass 元件 (shell 命令)

1. 使用配方和成品的子資料夾為您的元件建立資料夾。在 Greengrass 核心裝置上執行下列命令，以建立這些資料夾並變更為元件資料夾。將 `~/greengrassv2` 或 `%USERPROFILE%\greengrassv2` 取代為資料夾的路徑，以用於本機開發。

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. 使用文字編輯器來建立配方檔案，定義元件的中繼資料、參數、相依性、生命週期和平台功能。在配方檔案名稱中包含元件版本，以便您可以識別哪個配方反映哪個元件版本。您可以為您的配方選擇 YAML 或 JSON 格式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

JSON

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

YAML

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

Note

AWS IoT Greengrass 使用元件的語意版本。語意版本遵循 major.minor.patch 號碼系統。例如，版本1.0.0代表元件的第一個主要版本。如需詳細資訊，請參閱[語意版本規格](#)。

3. 定義元件的配方。如需詳細資訊，請參閱[AWS IoT Greengrass 元件配方參考](#)。

您的配方可能看起來類似下列 Hello World 範例配方。

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}"

```

此配方會執行 Hello World Python 指令碼，看起來可能會類似下列範例指令碼。

```
import sys

message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

4. 為要開發的元件版本建立資料夾。建議您針對每個元件版本的成品使用個別資料夾，以便識別每個元件版本適用的成品。執行下列命令。

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

您必須針對成品資料夾路徑使用下列格式。包含您在配方中指定的元件名稱和版本。

```
artifacts/componentName/componentVersion/
```

5. 在您在上一個步驟中建立的 資料夾中，為元件建立成品。成品可以包含軟體、映像和元件使用的任何其他二進位檔。

當您的元件準備就緒時，請[測試您的元件](#)。

使用本機部署測試 AWS IoT Greengrass 元件

如果您在核心裝置上開發 Greengrass 元件，您可以建立本機部署來安裝和測試它。遵循本節中的步驟來建立本機部署。

如果您在不同的電腦上開發元件，例如本機開發電腦，則無法建立本機部署。相反地，請將元件發佈至 AWS IoT Greengrass 服務，以便您可以將其部署至 Greengrass 核心裝置進行測試。如需詳細資訊，請參閱 [發佈元件以部署到您的核心裝置](#) 和 [將 AWS IoT Greengrass 元件部署至裝置](#)。

在 Greengrass 核心裝置上測試元件

1. 核心裝置會記錄事件，例如元件更新。您可以檢視此日誌檔案，以探索元件的錯誤並進行故障診斷，例如無效的配方。此日誌檔案也會顯示您的元件列印到標準輸出 (stdout) 的訊息。建議您在核心裝置上開啟額外的終端機工作階段，以即時觀察新的日誌訊息。開啟新的終端機工作階段，例如透過 SSH，並執行下列命令來檢視日誌。`/greengrass/v2` 將取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

您也可以檢視元件的日誌檔案。

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. 在原始終端機工作階段中，執行下列命令以使用 元件更新核心裝置。`/greengrass/v2` 將取代為 AWS IoT Greengrass 根資料夾的路徑，並將 `~/greengrassv2` 取代為本機開發資料夾的路徑。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \  
  --recipeDir ~/greengrassv2/recipes \  
  --artifactDir ~/greengrassv2/artifacts \  
  --componentName com.example.HelloWorld
```

```
--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

您也可以使用 `greengrass-cli deployment create` 命令來設定元件組態參數的值。如需詳細資訊，請參閱[建立](#)。

3. 使用 `greengrass-cli deployment status` 命令來監控元件部署的進度。

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. 在 Greengrass 核心裝置上執行元件時進行測試。當您完成此版本的元件時，您可以將其上傳至 AWS IoT Greengrass 服務。然後，您可以將元件部署到其他核心裝置。如需詳細資訊，請參閱[發佈元件以部署到您的核心裝置](#)。

發佈元件以部署到您的核心裝置

建置或完成元件版本後，您可以將其發佈至 AWS IoT Greengrass 服務。然後，您可以將它部署到 Greengrass 核心裝置。

如果您使用 [Greengrass 開發套件 CLI \(GDK CLI\)](#) 來[開發和建置元件](#)，您可以使用 [GDK CLI](#) 將元件發佈到 AWS 雲端。否則，[請使用內建 shell 命令和 AWS CLI](#) 來發佈元件。

您也可以使用從範本 AWS CloudFormation 建立元件和其他 AWS 資源。如需詳細資訊，請參閱 AWS CloudFormation 使用者指南[AWS::GreengrassV2::ComponentVersion](#)中的[什麼是 AWS CloudFormation ?](#) 和。

主題

- [發佈元件 \(GDK CLI\)](#)
- [發佈元件 \(shell 命令\)](#)

發佈元件 (GDK CLI)

遵循本節中的指示，使用 GDK CLI 發佈元件。GDK CLI 會將建置成品上傳至 S3 儲存貯體、更新配方中的成品 URIs，以及從配方建立元件。您可以指定要在 [GDK CLI 組態檔案中](#)使用的 S3 儲存貯體和區域。

如果您使用 GDK CLI 1.1.0 版或更新版本，您可以指定 `--bucket` 引數來指定 GDK CLI 上傳元件成品的 S3 儲存貯體。如果您未指定此引數，GDK CLI 會上傳至名為 `greengrass-bucket-region-accountId` 的 S3 儲存貯體，其中儲存#體和##是您在 `gdk-config.json` 中指定的值，而 `accountId` 是您的 AWS 帳戶 ID。如果儲存貯體不存在，GDK CLI 會建立儲存貯體。

Important

根據預設，核心裝置角色不允許存取 S3 儲存貯體。如果這是您第一次使用此 S3 儲存貯體，您必須將許可新增至角色，以允許核心裝置從此 S3 儲存貯體擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

發佈 Greengrass 元件 (GDK CLI)

1. 在命令提示字元或終端機中開啟元件資料夾。
2. 如果您尚未建立 Greengrass 元件，請建置 元件。[元件建置](#)命令會在元件greengrass-build資料夾中產生配方和成品。執行下列命令。

```
gdk component build
```

3. 將元件發佈至 AWS 雲端。[元件發佈](#)命令會將元件的成品上傳至 Amazon S3，並使用每個成品的 URI 更新元件的配方。然後，它會在 AWS IoT Greengrass 服務中建立元件。

Note

AWS IoT Greengrass 當您建立元件時，會計算每個成品的摘要。這表示在建立元件之後，您無法修改 S3 儲存貯體中的成品檔案。如果您這樣做，包含此元件的部署將會失敗，因為檔案摘要不相符。如果您修改成品檔案，則必須建立新的元件版本。

如果您NEXT_PATCH為 GDK CLI 組態檔案中的元件版本指定，則 GDK CLI 會使用 AWS IoT Greengrass 服務中尚不存在的下一個修補程式版本。

執行下列命令。

```
gdk component publish
```

輸出會告訴您 GDK CLI 建立的元件版本。

發佈元件之後，您可以將元件部署到核心裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

發佈元件 (shell 命令)

使用下列程序，使用 shell 命令和 AWS Command Line Interface () 發佈元件AWS CLI。當您發佈元件時，請執行下列動作：

1. 將元件成品發佈至 S3 儲存貯體。
2. 將每個成品的 Amazon S3 URI 新增至元件配方。
3. AWS IoT Greengrass 從元件配方在 中建立元件版本。

Note

您上傳的每個元件版本都必須是唯一的。請確定您上傳正確的元件版本，因為上傳之後就無法編輯。

您可以依照下列步驟，從您的開發電腦或 Greengrass 核心裝置發佈元件。

發佈元件 (shell 命令)

1. 如果元件使用 AWS IoT Greengrass 服務中存在的版本，則必須變更元件的版本。在文字編輯器中開啟配方、增加版本，並儲存檔案。選擇反映您對元件所做的變更的新版本。

Note

AWS IoT Greengrass 使用元件的語意版本。語意版本遵循 major.minor.patch 號碼系統。例如，版本 1.0.0 代表元件的第一個主要版本。如需詳細資訊，請參閱[語意版本規格](#)。

2. 如果您的元件有成品，請執行下列動作：
 - a. 將元件的成品發佈到 中的 S3 儲存貯體 AWS 帳戶。

Tip

我們建議您在 S3 儲存貯體成品路徑中包含元件名稱和版本。此命名機制可協助您維護先前元件版本使用的成品，因此您可以繼續支援先前的元件版本。

執行下列命令，將成品檔案發佈至 S3 儲存貯體。將 `amzn-s3-demo-bucket` 取代為儲存貯體的名稱，並將 `artifacts/com.example.HelloWorld/1.0.0/artifact.py` 取代為成品檔案的路徑。

```
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

根據預設，核心裝置角色不允許存取 S3 儲存貯體。如果這是您第一次使用此 S3 儲存貯體，您必須將許可新增至角色，以允許核心裝置從此 S3 儲存貯體擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

- b. 如果沒有名為 `artifacts` 的清單，請 Artifacts 將其新增至元件配方。Artifacts 清單會出現在每個資訊清單中，定義元件在其支援的每個平台上的需求（或元件對所有平台的預設需求）。
- c. 將每個成品新增至成品清單，或更新現有成品的 URI。Amazon S3 URI 由儲存貯體名稱和儲存貯體中成品物件的路徑組成。成品的 Amazon S3 URIs 看起來應該類似下列範例。

```
s3://amzn-s3-demo-bucket/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

完成這些步驟後，您的配方應該會有如下所示的 Artifacts 清單。

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
        {
          "URI": "s3://amzn-s3-demo-bucket/artifacts/  
MyGreengrassComponent/1.0.0/artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
  ]
}
```

Note

您可以新增 ZIP 成品 "Unarchive": "ZIP" 選項，以設定 AWS IoT Greengrass 核心軟體在元件部署時解壓縮成品。

YAML

```
...
Manifests:
  - Lifecycle:
    ...
    Artifacts:
      - URI: s3://amzn-s3-demo-bucket/artifacts/MyGreengrassComponent/1.0.0/
artifact.py
      Unarchive: NONE
```

Note

您可以使用 Unarchive: ZIP 選項來設定 AWS IoT Greengrass 核心軟體，以在元件部署時解壓縮 ZIP 成品。如需如何在元件中使用 ZIP 成品的詳細資訊，請參閱 [artifacts : decompressedPath 配方變數](#)。

如需配方的詳細資訊，請參閱 [AWS IoT Greengrass 元件配方參考](#)。

3. 使用 AWS IoT Greengrass 主控台從配方檔案建立元件。

執行下列命令，從配方檔案建立元件。此命令會建立元件，並將其發佈為 中的私有 AWS IoT Greengrass 元件 AWS 帳戶。將 *path/to/recipeFile* 取代為配方檔案的路徑。

```
aws greengrassv2 create-component-version --inline-recipe fileb://path/to/
recipeFile
```

arn 從回應中複製，以在下一個步驟中檢查元件的狀態。

 Note

AWS IoT Greengrass 當您建立元件時，會計算每個成品的摘要。這表示在建立元件之後，您無法修改 S3 儲存貯體中的成品檔案。如果您這樣做，包含此元件的部署將會失敗，因為檔案摘要不相符。如果您修改成品檔案，則必須建立新的元件版本。

4. AWS IoT Greengrass 服務中的每個元件都有狀態。執行下列命令以確認您在此程序中發佈的元件版本狀態。將 `com.example.HelloWorld` 和 `1.0.0` 取代為要查詢的元件版本。將 `arn` 為上一個步驟的 ARN。

```
aws greengrassv2 describe-component --arn "arn:aws:greengrass:region:account-id:components:com.example.HelloWorld:versions:1.0.0"
```

操作會傳回包含元件中繼資料的回應。中繼資料包含 `status` 物件，其中包含元件狀態和任何錯誤，如適用。

當元件狀態為 `DEPLOYABLE`，您可以將元件部署到裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

與 AWS 服務互動

Greengrass 核心裝置使用 X.509 憑證，AWS IoT Core 使用 TLS 相互身分驗證通訊協定連線至。這些憑證可讓裝置 AWS IoT 在沒有 AWS 登入資料的情況下與互動，通常包含存取金鑰 ID 和私密存取金鑰。其他服務 AWS 需要 AWS 登入資料，而不是 X.509 憑證，才能呼叫服務端點的 API 操作。AWS IoT Core 具有登入資料提供者，可讓裝置使用其 X.509 憑證來驗證 AWS 請求。AWS IoT 登入資料提供者會使用 X.509 憑證來驗證裝置，並以臨時、有限權限的安全字符的形式發出 AWS 登入資料。裝置可以使用此字符來簽署和驗證任何 AWS 請求。這不需要在 Greengrass 核心裝置上存放 AWS 登入資料。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權對 AWS 服務的直接呼叫](#)。

若要從中擷取登入資料 AWS IoT，Greengrass 核心裝置會使用指向 IAM AWS IoT 角色的角色別名。此 IAM 角色稱為字符交換角色。您可以在安裝 AWS IoT Greengrass Core 軟體時建立角色別名和字符交換角色。若要指定核心裝置使用的角色別名，請設定的 `iotRoleAlias` 參數[Greengrass 核](#)。

AWS IoT 登入資料提供者會代表您擔任權杖交換角色，以提供 AWS 登入資料給核心裝置。您可以將適當的 IAM 政策連接到此角色，以允許核心裝置存取您的 AWS 資源，例如 S3 儲存貯體中的元件成品。如需如何設定字符交換角色的詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

Greengrass 核心裝置會將 AWS 登入資料存放在記憶體中，而且登入資料預設會在一小時後過期。如果 AWS IoT Greengrass 核心軟體重新啟動，則必須再次擷取登入資料。您可以使用 [UpdateRoleAlias](#) 操作來設定憑證有效的持續時間。

AWS IoT Greengrass 提供公有元件，也就是權杖交換服務元件，您可以在自訂元件中將其定義為與 AWS 服務互動的相依性。權杖交換服務為您的元件提供環境變數 `AWS_CONTAINER_CREDENTIALS_FULL_URI`，此變數會將 URI 定義為提供 AWS 憑證的本機伺服器。當您建立 AWS SDK 用戶端時，用戶端會檢查此環境變數，並連線至本機伺服器以擷取 AWS 登入資料，並使用它們來簽署 API 請求。這可讓您使用 AWS SDKs 和其他工具來呼叫元件中的 AWS 服務。如需詳細資訊，請參閱[字交換服務](#)。

Important

2016 年 7 月 13 日，AWS SDKs 已新增以這種方式取得 AWS 憑證的支援。您的元件必須使用在該日期或之後建立的 AWS SDK 版本。如需詳細資訊，請參閱《Amazon Elastic Container Service 開發人員指南》中的[使用支援的 AWS SDK](#)。

若要在自訂元件中取得 AWS 登入資料，請在元件配方中定義 `aws.greengrass.TokenExchangeService` 做為相依性。下列範例配方會定義安裝 [boto3](#) 並執行 Python 指令碼的元件，該指令碼使用字交換服務的 AWS 登入資料來列出 Amazon S3 儲存貯體。

Note

若要執行此範例元件，您的裝置必須具有 `s3:ListAllMyBuckets` 許可。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

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
```

此範例元件會執行下列 Python 指令碼，list_s3_buckets.py 其中會列出 Amazon S3 儲存貯體。

```
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)
```

執行 Docker 容器

您可以將 AWS IoT Greengrass 元件設定為從存放在下列位置的映像執行 [Docker](#) 容器：

- Amazon Elastic Container Registry (Amazon ECR) 中的公有和私有映像儲存庫
- 公有 Docker Hub 儲存庫
- 公有 Docker 信任的登錄檔

- S3 bucket (S3 儲存貯體)

在您的自訂元件中，包含 Docker 映像 URI 做為擷取映像並在核心裝置上執行的成品。對於 Amazon ECR 和 Docker Hub 映像，您可以使用 [Docker 應用程式管理員](#) 元件來下載映像並管理私有 Amazon ECR 儲存庫的登入資料。

主題

- [要求](#)
- [從 Amazon ECR 或 Docker Hub 中的公有映像執行 Docker 容器](#)
- [從 Amazon ECR 中的私有映像執行 Docker 容器](#)
- [從 Amazon S3 中的映像執行 Docker 容器](#)
- [在 Docker 容器元件中使用程序間通訊](#)
- [在 Docker 容器元件 \(Linux\) 中使用 AWS 登入資料](#)
- [在 Docker 容器元件 \(Linux\) 中使用串流管理員](#)

要求

若要在元件中執行 Docker 容器，您需要下列項目：

- Greengrass 核心裝置。如果您沒有帳戶，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。
- [Docker 引擎](#) 1.9.1 或更新版本安裝在 Greengrass 核心裝置上。20.10 版是經過驗證可與 AWS IoT Greengrass Core 軟體搭配使用的最新版本。您必須先直接在核心裝置上安裝 Docker，才能部署執行 Docker 容器的元件。

Tip

您也可以設定核心裝置，以在元件安裝時安裝 Docker Engine。例如，下列安裝指令碼會在載入 Docker 映像之前安裝 Docker 引擎。此安裝指令碼適用於 Debian 型 Linux 發行版本，例如 Ubuntu。如果您設定元件使用此命令安裝 Docker Engine，您可能需要在生命週期指令碼 `true` 中 `RequiresPrivilege` 將設定為 `sudo`，才能執行安裝和 `docker` 命令。如需詳細資訊，請參閱 [AWS IoT Greengrass 元件配方參考](#)。

```
apt-get install docker-ce docker-ce-cli containerd.io && docker load -i  
{artifacts:path}/hello-world.tar
```


- 執行 Docker 容器元件的系統使用者必須具有根或管理員許可，或者您必須將 Docker 設定為以非根或非註冊使用者身分執行。
- 在 Linux 裝置上，您可以將使用者新增至 docker 群組，在沒有 `sudo` 的情況下呼叫 docker 命令。
- 在 Windows 裝置上，您可以將使用者新增至 docker-users 群組，以呼叫沒有管理員權限的 docker 命令。

Linux or Unix

若要將 `ggc_user` 或您用來執行 Docker 容器元件的非根使用者新增至 docker 群組，請執行下列命令。

```
sudo usermod -aG docker ggc_user
```

如需詳細資訊，請參閱[以非根使用者的身管理 Docker](#)。

Windows Command Prompt (CMD)

若要將 `ggc_user` 或您用來執行 Docker 容器元件的使用者新增至 docker-users 群組，請以管理員身分執行下列命令。

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

若要將 `ggc_user` 或您用來執行 Docker 容器元件的使用者新增至 docker-users 群組，請以管理員身分執行下列命令。

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Docker 容器元件存取的檔案會掛載為 [Docker 容器中的磁碟區](#)。
- 如果您將 [AWS IoT Greengrass Core 軟體設定為使用網路代理](#)，則必須將 [Docker 設定為使用相同的代理伺服器](#)。

除了這些要求之外，如果適用於您的環境，您還必須符合下列要求：

- 若要使用 [Docker Compose](#) 建立和啟動 Docker 容器，請在 Greengrass 核心裝置上安裝 Docker Compose，並將 Docker Compose 檔案上傳至 S3 儲存貯體。您必須將 Compose 檔案存放在與元

件相同的 S3 AWS 帳戶 AWS 區域 儲存貯體中。如需在自訂元件中使用 `docker-compose up` 命令的範例，請參閱 [從 Amazon ECR 或 Docker Hub 中的公有映像執行 Docker 容器](#)。

- 如果您在網路代理 AWS IoT Greengrass 之後執行，請將 Docker 協助程式設定為使用[代理伺服器](#)。
- 如果您的 Docker 映像存放在 Amazon ECR 或 Docker Hub 中，請在 [Docker 容器元件中包含 Docker 元件管理員](#) 元件做為相依性。您必須先在核心裝置上啟動 Docker 協助程式，才能部署元件。

此外，請包含影像 URIs 做為元件成品。映像 URIs 的格式必須 `docker:registry/image[:tag/@digest]` 如下列範例所示：

- 私有 Amazon ECR 映像：`docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag/@digest]`
- 公有 Amazon ECR 映像：`docker:public.ecr.aws/repository/image[:tag/@digest]`
- 公有 Docker Hub 映像：`docker:name[:tag/@digest]`

如需從存放在公有儲存庫中的映像執行 Docker 容器的詳細資訊，請參閱 [從 Amazon ECR 或 Docker Hub 中的公有映像執行 Docker 容器](#)。

- 如果您的 Docker 映像存放在 Amazon ECR 私有儲存庫中，則必須在 Docker 容器元件中包含字符交換服務元件做為相依性。此外，[Greengrass 裝置角色](#) 必須允許 `ecr:GetAuthorizationToken`、`ecr:BatchGetImage` 和 `ecr:GetDownloadUrlForLayer` 動作，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

```
]
}
```

如需有關從存放在 Amazon ECR 私有儲存庫中的映像執行 Docker 容器的資訊，請參閱 [從 Amazon ECR 中的私有映像執行 Docker 容器](#)。

- 若要使用存放在 Amazon ECR 私有儲存庫中的 Docker 映像，私有儲存庫必須與核心裝置 AWS 區域 位於相同的 中。
- 如果您的 Docker 映像或 Compose 檔案存放在 S3 儲存貯體中，[Greengrass 裝置角色](#)必須允許 s3:GetObject 許可，以允許核心裝置將映像下載為元件成品，如下列範例 IAM 政策所示。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

如需有關從存放在 Amazon S3 中的映像執行 Docker 容器的資訊，請參閱 [從 Amazon S3 中的映像執行 Docker 容器](#)。

- 若要在 Docker 容器元件中使用程序間通訊 (IPC) AWS、登入資料或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：
- [在 Docker 容器元件中使用程序間通訊](#)
- [在 Docker 容器元件 \(Linux\) 中使用 AWS 登入資料](#)
- [在 Docker 容器元件 \(Linux\) 中使用串流管理員](#)

從 Amazon ECR 或 Docker Hub 中的公有映像執行 Docker 容器

本節說明如何建立使用 Docker Compose 從存放 Amazon ECR 和 Docker Hub 的 Docker 映像檔執行 Docker 容器的自訂元件。

使用 Docker Compose 執行 Docker 容器

1. 建立 Docker Compose 檔案並將其上傳至 Amazon S3 儲存貯體。確定 [Greengrass 裝置角色](#) 允許 `s3:GetObject` 許可，讓裝置存取 Compose 檔案。下列範例中顯示的 Compose 檔案範例包括來自 Amazon ECR 的 Amazon CloudWatch Agent 映像和來自 Docker Hub 的 MySQL 映像。

```
version: "3"
services:
  cloudwatchagent:
    image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
  mysql:
    image: "mysql:8.0"
```

2. 在 AWS IoT Greengrass 核心裝置上[建立自訂元件](#)。下列範例中顯示的範例配方案具有下列屬性：
 - Docker 應用程式管理員元件做為相依性。此元件可讓從公 AWS IoT Greengrass 有 Amazon ECR 和 Docker Hub 儲存庫下載映像。
 - 元件成品，指定公有 Amazon ECR 儲存庫中的 Docker 映像。
 - 元件成品，指定公有 Docker Hub 儲存庫中的 Docker 映像。
 - 元件成品，指定包含您要執行之 Docker 映像容器的 Docker Compose 檔案。
 - 生命週期執行指令碼，使用 [docker-compose up](#) 從指定的映像建立和啟動容器。

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

若要在 Docker 容器元件中使用程序間通訊 (IPC) AWS、登入資料或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：

- [在 Docker 容器元件中使用程序間通訊](#)
- [在 Docker 容器元件 \(Linux\) 中使用 AWS 登入資料](#)
- [在 Docker 容器元件 \(Linux\) 中使用串流管理員](#)

3. [測試元件](#)以驗證其是否如預期般運作。

Important

您必須先安裝和啟動 Docker 協助程式，才能部署元件。

在本機部署元件之後，您可以執行 [Docker 容器 ls](#) 命令，以驗證您的容器是否執行。

```
docker container ls
```

4. 當元件準備就緒時，將元件上傳至 AWS IoT Greengrass 以部署到其他核心裝置。如需詳細資訊，請參閱[發佈元件以部署到您的核心裝置](#)。

從 Amazon ECR 中的私有映像執行 Docker 容器

本節說明如何建立自訂元件，從存放在 Amazon ECR 中私有儲存庫中的 Docker 映像執行 Docker 容器。

執行 Docker 容器

1. 在 AWS IoT Greengrass 核心裝置上[建立自訂元件](#)。使用下列範例配方，其具有下列屬性：
 - Docker 應用程式管理員元件做為相依性。此元件可讓 AWS IoT Greengrass 管理登入資料，以從私有儲存庫下載映像。

- 權杖交換服務元件做為相依性。此元件可讓 AWS IoT Greengrass 擷取 AWS 登入資料以與 Amazon ECR 互動。
- 元件成品，指定私有 Amazon ECR 儲存庫中的 Docker 映像。
- 生命週期執行指令碼，使用 [docker run](#) 從映像建立和啟動容器。

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

若要在 Docker 容器元件中使用程序間通訊 (IPC) AWS、登入資料或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：

- [在 Docker 容器元件中使用程序間通訊](#)
- [在 Docker 容器元件 \(Linux\) 中使用 AWS 登入資料](#)
- [在 Docker 容器元件 \(Linux\) 中使用串流管理員](#)

2. [測試元件](#)以驗證其是否如預期般運作。

Important

您必須先安裝和啟動 Docker 協助程式，才能部署元件。

在本機部署元件之後，您可以執行 [Docker 容器 ls](#) 命令，以驗證您的容器是否執行。

```
docker container ls
```

3. 將元件上傳至 AWS IoT Greengrass 以部署至其他核心裝置。如需詳細資訊，請參閱[發佈元件以部署到您的核心裝置](#)。

從 Amazon S3 中的映像執行 Docker 容器

本節說明如何從存放在 Amazon S3 的 Docker 映像執行 Docker 容器。

從 Amazon S3 中的映像執行 Docker 容器

1. 執行 [docker save](#) 命令來建立 Docker 容器的備份。您提供此備份做為執行容器的元件成品 AWS IoT Greengrass。將 *hello-world* 取代為映像的名稱，並將 *hello-world.tar* 取代為要建立的封存檔案名稱。

```
docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
```

2. 在 AWS IoT Greengrass 核心裝置上[建立自訂元件](#)。使用下列範例配方，其具有下列屬性：
 - 使用 [Docker 載入](#) 從封存取入 Docker 映像的生命週期安裝指令碼。
 - 生命週期執行指令碼，使用 [docker run](#) 從映像建立和啟動容器。--rm 選項會在容器結束時清除容器。

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

若要在 Docker 容器元件中使用程序間通訊 (IPC) AWS、登入資料或串流管理員，您必須在執行 Docker 容器時指定其他選項。如需詳細資訊，請參閱下列內容：

- [在 Docker 容器元件中使用程序間通訊](#)
- [在 Docker 容器元件 \(Linux\) 中使用 AWS 登入資料](#)
- [在 Docker 容器元件 \(Linux\) 中使用串流管理員](#)

3. [測試元件](#)以驗證其是否如預期般運作。

在本機部署元件之後，您可以執行 [Docker 容器 ls](#) 命令，以驗證您的容器是否執行。

```
docker container ls
```

4. 當元件就緒時，將 Docker 映像封存上傳至 S3 儲存貯體，然後將其 URI 新增至元件配方。然後，您可以將元件上傳至 AWS IoT Greengrass 以部署至其他核心裝置。如需詳細資訊，請參閱[發佈元件以部署到您的核心裝置](#)。

完成後，元件配方應如下所示。

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
```

在 Docker 容器元件中使用程序間通訊

您可以使用 `Greengrass` 處理程序間通訊 (IPC) 程式庫 `AWS IoT Device SDK`，與 `Greengrass` 核、其他 `Greengrass` 元件和進行通訊 `AWS IoT Core`。如需詳細資訊，請參閱[使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#)。

若要在 Docker 容器元件中使用 IPC，您必須使用下列參數執行 Docker 容器：

- 在容器中掛載 IPC 通訊端。`Greengrass` 核提供 `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` 環境變數中的 IPC 通訊端檔案路徑。
- 將 `SVCUID` 和 `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` 環境變數設定為 `Greengrass` 核提供給元件的值。您的元件使用這些環境變數來驗證與 `Greengrass` 核的連線。

Example 範例配方：發佈 MQTT 訊息至 `AWS IoT Core` (Python)

下列配方定義發佈 MQTT 訊息的 Docker 容器元件範例 `AWS IoT Core`。此配方具有下列屬性：

- 允許元件在所有主題 AWS IoT Core 上發佈 MQTT 訊息至 的授權政策 (accessControl)。如需詳細資訊，請參閱 [授權元件執行 IPC 操作](#) 和 [AWS IoT Core MQTT IPC 授權](#)。
- 元件成品，指定 Docker 影像做為 Amazon S3 中的 TAR 封存。
- 生命週期安裝指令碼，可從 TAR 封存檔載入 Docker 映像。
- 從映像執行 Docker 容器的生命週期執行指令碼。[Docker run](#) 命令具有下列引數：
 - -v 引數會將 Greengrass IPC 通訊端掛載在容器中。
 - 前兩個 -e 引數會在 Docker 容器中設定所需的環境變數。
 - 其他 -e 引數會設定此範例所使用的環境變數。
 - --rm 引數會在容器結束時清除容器。

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
```

在 Docker 容器元件 (Linux) 中使用 AWS 登入資料

您可以使用[字符交換服務元件](#)與 Greengrass 元件中的 AWS 服務互動。此元件會使用本機容器伺服器，從核心裝置的[字符交換角色](#)提供 AWS 登入資料。如需詳細資訊，請參閱[與 AWS 服務互動](#)。

Note

本節中的範例僅適用於 Linux 核心裝置。

若要在 Docker 容器元件中使用字符交換服務的 AWS 憑證，您必須使用下列參數執行 Docker 容器：

- 使用 `--network=host` 引數提供主機網路的存取權。此選項可讓 Docker 容器連線至本機字符交換服務，以擷取 AWS 憑證。此引數僅適用於 Docker for Linux。

⚠ Warning

此選項可讓容器存取主機上的所有本機網路介面，因此此選項比沒有主機網路存取權的情況下執行 Docker 容器更不安全。當您開發和執行使用此選項的 Docker 容器元件時，請考慮這一點。如需詳細資訊，請參閱 Docker 文件中的[網路：主機](#)。

- 將 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 和 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 環境變數設定為 Greengrass 核提供給 components 的值。AWS SDKs 使用這些環境變數來擷取 AWS 登入資料。

Example 範例配方：列出 Docker 容器元件中的 S3 儲存貯體 (Python)

下列配方會定義範例 Docker 容器元件，列出 中的 S3 儲存貯體 AWS 帳戶。此配方具有下列屬性：

- 做為相依性的字符交換服務元件。此相依性可讓 元件擷取 AWS 登入資料，以與其他 AWS 服務互動。
- 元件成品，指定 Docker 影像做為 Amazon S3 中的 tar 封存。
- 生命週期安裝指令碼，可從 TAR 封存檔載入 Docker 映像。
- 從映像執行 Docker 容器的生命週期執行指令碼。[Docker run](#) 命令具有下列引數：
 - `--network=host` 引數提供容器對主機網路的存取權，讓容器可以連線至字符交換服務。
 - `-e` 引數會在 Docker 容器中設定所需的環境變數。
 - `--rm` 引數會在容器結束時清除容器。

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
```

在 Docker 容器元件 (Linux) 中使用串流管理員

您可以使用[串流管理員元件](#)來管理 Greengrass 元件中的資料串流。此元件可讓您處理資料串流並將大量 IoT 資料傳輸至 AWS 雲端。AWS IoT Greengrass 提供串流管理員 SDK，讓您用來與串流管理員元件互動。如需詳細資訊，請參閱[管理 Greengrass 核心裝置上的資料串流](#)。

Note

本節中的範例僅適用於 Linux 核心裝置。

若要在 Docker 容器元件中使用串流管理員 SDK，您必須使用下列參數執行 Docker 容器：

- 使用 `--network=host` 引數提供主機網路的存取權。此選項可讓 Docker 容器透過本機 TLS 連線與串流管理員元件互動。此引數僅適用於適用於 Linux 的 Docker

Warning

此選項可讓容器存取主機上的所有本機網路介面，因此此選項比沒有主機網路存取權的情況下執行 Docker 容器更不安全。當您開發和執行使用此選項的 Docker 容器元件時，請考慮這一點。如需詳細資訊，請參閱 Docker 文件中的[網路：主機](#)。

- 如果您將串流管理員元件設定為需要身分驗證，這是預設行為，請將 `AWS_CONTAINER_CREDENTIALS_FULL_URI` 環境變數設定為 Greengrass 核提供給元件的值。如需詳細資訊，請參閱[串流管理員組態](#)。
- 如果您將串流管理員元件設定為使用非預設連接埠，請使用[程序間通訊 \(IPC\)](#) 從串流管理員元件組態取得連接埠。您必須使用其他選項來執行 Docker 容器，才能使用 IPC。如需詳細資訊，請參閱下列內容：
 - [連線至應用程式程式碼中的串流管理員](#)
 - [在 Docker 容器元件中使用程序間通訊](#)

Example 範例配方：將檔案串流到 Docker 容器元件 (Python) 中的 S3 儲存貯體

下列配方定義範例 Docker 容器元件，可建立檔案並將其串流至 S3 儲存貯體。此配方案具有下列屬性：

- 串流管理員元件做為相依性。此相依性可讓元件使用串流管理員 SDK 與串流管理員元件互動。
- 元件成品，指定 Docker 影像做為 Amazon S3 中的 TAR 封存。
- 生命週期安裝指令碼，可從 TAR 封存檔載入 Docker 映像。
- 從映像執行 Docker 容器的生命週期執行指令碼。[Docker run](#) 命令具有下列引數：
 - `--network=host` 引數提供容器對主機網路的存取權，讓容器可以連線至串流管理員元件。
 - 第一個 `-e` 引數會在 Docker 容器中設定所需的 `AWS_CONTAINER_AUTHORIZATION_TOKEN` 環境變數。
 - 其他 `-e` 引數會設定此範例所使用的環境變數。
 - `-v` 引數會將元件的[工作資料夾](#)掛載在容器中。此範例會在工作資料夾中建立 檔案，以使用串流管理員將該檔案上傳至 Amazon S3。
 - `--rm` 引數會在容器結束時清除容器。

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 元件配方參考

元件配方是定義元件詳細資訊、相依性、成品和生命週期的檔案。例如，元件生命週期會指定要執行的命令來安裝、執行和關閉元件。AWS IoT Greengrass 核心使用您在配方中定義的生命週期來安裝和執行元件。當您部署元件時，AWS IoT Greengrass 此服務會使用配方來識別要部署至核心裝置的相依性和成品。

在配方中，您可以為元件支援的每個平台定義唯一的相依性和生命週期。您可以使用此功能，將元件部署到具有不同需求的多個平台的裝置。您也可以使用此功能來 AWS IoT Greengrass 防止 在不支援它的裝置上安裝元件。

每個配方都包含資訊清單清單。每個資訊清單都會指定一組平台需求，以及用於其平台符合這些需求之核心裝置的生命週期和成品。核心裝置使用第一個資訊清單，其中包含裝置符合的平台需求。指定沒有任何平台需求的資訊清單，以符合任何核心裝置。

您也可以指定不在資訊清單中的全域生命週期。在 全域生命週期中，您可以使用選取索引鍵來識別生命週期的子區段。然後，您可以在資訊清單中指定這些選取索引鍵，以在資訊清單的生命週期之外，使用全域生命週期的這些區段。只有在資訊清單未定義生命週期時，核心裝置才會使用資訊清單的選擇索引鍵。您可以使用資訊清單中的all選取項目來比對全域生命週期的區段，無需選取索引鍵。

AWS IoT Greengrass 核心軟體選取符合核心裝置的資訊清單後，會執行下列動作來識別要使用的生命週期步驟：

- 如果選取的資訊清單定義生命週期，核心裝置會使用該生命週期。
- 如果選取的資訊清單未定義生命週期，核心裝置會使用全域生命週期。核心裝置會執行下列動作，以識別要使用的全域生命週期區段：
 - 如果資訊清單定義選取索引鍵，核心裝置會使用包含資訊清單選取索引鍵的全域生命週期區段。
 - 如果資訊清單未定義選取索引鍵，核心裝置會使用沒有選取索引鍵的全域生命週期區段。此行為等同於定義all選取的資訊清單。

⚠ Important

核心裝置必須符合至少一個資訊清單的平台需求，才能安裝元件。如果沒有資訊清單符合核心裝置，則 AWS IoT Greengrass Core 軟體不會安裝元件，且部署會失敗。

您可以 [JSON](#) 或 [YAML](#) 格式定義配方。配方範例區段包含每種格式的配方。

主題

- [配方驗證](#)
- [配方格式](#)
- [配方變數](#)
- [配方範例](#)

配方驗證

Greengrass 會在建立元件版本時驗證 JSON 或 YAML 元件配方。此配方驗證會檢查您的 JSON 或 YAML 元件配方是否有常見錯誤，以防止潛在的部署問題。驗證會檢查配方是否有常見錯誤（例如缺少逗號、括號和欄位），並確保配方格式正確。

如果您收到配方驗證錯誤訊息，請檢查您的配方是否有任何遺漏的逗號、括號或欄位。查看[配方格式](#)，確認您未遺漏任何欄位。

配方格式

當您定義元件的配方時，您可以在配方文件中指定下列資訊。相同的結構適用於 YAML 和 JSON 格式的配方。

RecipeFormatVersion

配方的範本版本。選擇下列選項：

- 2020-01-25

ComponentName

此配方定義的元件名稱。元件名稱 AWS 帳戶 在每個區域中的 中必須是唯一的。

提示

- 使用反向網域名稱格式，以避免公司內發生名稱衝突。例如，如果您的公司擁有example.com並處理太陽能專案，您可以將 Hello World 元件命名為 com.example.solar.HelloWorld。這有助於避免貴公司內的元件名稱衝突。
- 避免在元件名稱中使用aws.greengrass字首。AWS IoT Greengrass 會針對其提供的[公有元件](#)使用此字首。如果您選擇與公有元件相同的名稱，您的元件會取代該元件。然後，在部署與該公有元件相依的元件時，AWS IoT Greengrass 提供您的元件而非公有元件。此功能可讓您覆寫公有元件的行為，但如果您不打算覆寫公有元件，也可能會破壞其他元件。

ComponentVersion

元件的版本。主要、次要和修補程式值的最大值為 999999。

Note

AWS IoT Greengrass 使用元件的語意版本。語意版本遵循 major.minor.patch 號碼系統。例如，版本1.0.0代表元件的第一個主要版本。如需詳細資訊，請參閱[語意版本規格](#)。

ComponentDescription

(選用) 元件的描述。

ComponentPublisher

元件的發佈者或作者。

ComponentConfiguration

(選用) 定義元件組態或參數的物件。您可以定義預設組態，然後在部署元件時，指定要提供給元件的組態物件。元件組態支援巢狀參數和結構。此物件包含下列資訊：

DefaultConfiguration

定義元件預設組態的物件。您可以定義此物件的結構。

Note

AWS IoT Greengrass 使用 JSON 做為組態值。JSON 會指定數字類型，但不會區分整數和浮點數。因此，組態值可能會轉換為浮點數 AWS IoT Greengrass。為了確保您的元件使用正確的資料類型，建議您將數值組態值定義為字串。然後，讓您的元件將它們剖析為整數或浮點數。這可確保您的組態值在組態和核心裝置上具有相同的類型。

ComponentDependencies

(選用) 物件的字典，每個物件都會定義元件的元件相依性。每個物件的 索引鍵可識別元件相依性的名稱。AWS IoT Greengrass 當元件安裝時，安裝元件相依性。在啟動元件之前 AWS IoT Greengrass 等待相依性啟動。每個物件都包含下列資訊：

VersionRequirement

定義此相依性的相容元件版本的 npm 樣式語意版本限制。您可以指定版本或一系列版本。如需詳細資訊，請參閱 [npm 語意版本計算器](#)。

DependencyType

(選用) 此相依性的類型。您可以從以下選項中選擇。

- SOFT - 在相依性變更狀態時，元件不會重新啟動。
- HARD - 在相依性變更狀態時，元件會重新啟動。

預設為 HARD。

ComponentType

(選用) 元件的類型。

Note

我們不建議您在 recipe 中指定元件類型。當您建立元件時，會為您 AWS IoT Greengrass 設定 類型。

類型可以是下列類型之一：

- `aws.greengrass.generic` – 元件會執行命令或提供成品。
- `aws.greengrass.lambda` – 元件會使用 Lambda [啟動器元件執行 Lambda](#) 函數。ComponentSource 參數指定此元件執行之 Lambda 函數的 ARN。

我們不建議您使用此選項，因為當您從 Lambda 函數建立元件 AWS IoT Greengrass 時，它會由設定。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

- `aws.greengrass.plugin` – 元件在與 Greengrass 核相同的 Java 虛擬機器 (JVM) 中執行。如果您部署或重新啟動外掛程式元件，Greengrass 核會重新啟動。

外掛程式元件使用與 Greengrass 核相同的日誌檔案。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

我們不建議您在元件配方中使用此選項，因為它適用於以 Java AWS 編寫且直接與 Greengrass 核界面的元件。如需哪些公有元件是外掛程式的詳細資訊，請參閱[AWS 提供的元件](#)。

- `aws.greengrass.nucleus` – 核心元件。如需詳細資訊，請參閱[Greengrass 核](#)。

我們不建議您在元件配方中使用此選項。它適用於 Greengrass 核元件，可提供 AWS IoT Greengrass Core 軟體的最低功能。

`aws.greengrass.generic` 當您從配方建立元件，或從 Lambda 函數建立元件 `aws.greengrass.lambda` 時，預設為。

如需詳細資訊，請參閱[元件類型](#)。

ComponentSource

(選用) 元件執行的 Lambda 函數 ARN。

我們不建議您在 recipe 中指定元件來源。當您從 Lambda 函數建立元件時，會為您 AWS IoT Greengrass 設定此參數。如需詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

Manifests

各定義元件生命週期、參數和平台需求的物件清單。如果核心裝置符合多個資訊清單的平台需求，AWS IoT Greengrass 會使用核心裝置符合的第一個資訊清單。為了確保核心裝置使用正確的資訊清單，請先定義具有更嚴格平台需求的資訊清單。適用於所有平台的資訊清單必須是清單中的最後一個資訊清單。

Important

核心裝置必須符合至少一個資訊清單的平台需求，才能安裝元件。如果沒有資訊清單符合核心裝置，則 AWS IoT Greengrass Core 軟體不會安裝元件，且部署會失敗。

每個物件都包含下列資訊：

Name

(選用) 此資訊清單定義之平台的易記名稱。

如果您省略此參數，會從平台 `os` 和 AWS IoT Greengrass 建立名稱 `architecture`。

Platform

(選用) 定義適用此資訊清單之平台的物件。省略此參數，以定義適用於所有平台的資訊清單。

此物件會指定核心裝置執行所在的平台的鍵/值對。當您部署此元件時，AWS IoT Greengrass 核心軟體會將這些鍵/值對與核心裝置上的平台屬性進行比較。AWS IoT Greengrass 核心軟體一律定義 `os` 和 `architecture`，且可能定義其他屬性。您可以在部署 Greengrass 核元件時指定核心裝置的自訂平台屬性。如需詳細資訊，請參閱 [平台覆寫 Greengrass 核元件的參數](#)。 ???

對於每個鍵/值對，您可以指定下列其中一個值：

- 確切的值，例如 `linux` 或 `windows`。確切值必須以字母或數字開頭。
- `*`，符合任何值。當值不存在時，這也符合。
- Java 樣式的規則表達式，例如 `/windows|linux/`。規則表達式必須以斜線字元 `()` 開頭和結尾。例如，規則表達式 `/.+ /` 符合任何非空白值。

此物件包含下列資訊：

runtime

此資訊清單支援的平台的 [Greengrass 核執行時間](#)。使用平台 定義多個資訊清單時 `runtime`，配方中支援的執行時間值僅為 `aws_nucleus_lite` 和 `*`。若要以傳統裝置為目標，不得在配方中指定執行時間欄位。支援的 Greengrass Nucleus 執行時間包括下列值：

- `*`
- `aws_nucleus_lite`

os

(選用) 此資訊清單支援的平台的作業系統名稱。常見的平台包括下列值：

- `linux`
- `windows`
- `darwin` (macOS)

architecture

(選用) 此資訊清單支援的平台處理器架構。常見的架構包括下列值：

- amd64
- arm
- aarch64
- x86

architecture.detail

(選用) 此資訊清單支援的平台處理器架構詳細資訊。常見的架構詳細資訊包括下列值：

- arm61
- arm71
- arm81

key

(選用) 您為此資訊清單定義的平台屬性。將`##`取代為平台屬性的名稱。Core AWS IoT Greengrass 軟體會比對此平台屬性與您在 Greengrass 核元件組態中指定的鍵值對。如需詳細資訊，請參閱 [平台覆寫 Greengrass 核元件的參數](#)。 ???

Tip

使用反向網域名稱格式，以避免公司內發生名稱衝突。例如，如果您的公司擁有example.com並處理無線電專案，您可以命名自訂平台屬性com.example.radio.RadioModule。這有助於避免您公司內的平台屬性名稱衝突。

例如，您可以定義平台屬性 com.example.radio.RadioModule，根據核心裝置上可用的無線電模組來指定不同的資訊清單。每個資訊清單可以包含套用至不同硬體組態的不同成品，以便您將最少的軟體集部署到核心裝置。

Lifecycle

物件或字串，定義如何在此資訊清單定義的平台上安裝和執行 元件。您也可以定義適用於所有平台的[全域生命週期](#)。只有在要使用的資訊清單未指定生命週期時，核心裝置才會使用全域生命週期。

 Note

您可以在資訊清單中定義此生命週期。您在此處指定的生命週期步驟僅適用於此資訊清單定義的平台。您也可以定義適用於所有平台的[全域生命週期](#)。

此物件或字串包含下列資訊：

Setenv

(選用) 要提供給所有生命週期指令碼的環境變數字典。您可以在每個生命週期指令碼Setenv中使用 覆寫這些環境變數。

install

(選用) 物件或字串，定義元件安裝時要執行的指令碼。AWS IoT Greengrass 核心軟體也會在每次軟體啟動時執行此生命週期步驟。

如果install指令碼以成功代碼結束，元件會進入 INSTALLED 狀態。

此物件或字串包含下列資訊：

Script

要執行的指令碼。

RequiresPrivilege

(選用) 您可以使用根權限執行指令碼。如果您將此選項設定為 true，則 AWS IoT Greengrass Core 軟體會執行此生命週期指令碼做為根目錄，而不是您設定為執行此元件的系統使用者。預設為 false。

Skipif

(選用) 檢查以判斷是否執行指令碼。您可以定義 來檢查可執行檔是否在路徑上，或檔案是否存在。如果輸出為 true，則 AWS IoT Greengrass Core 軟體會略過 步驟。選擇下列其中一個檢查：

- onpath *runnable* – 檢查可執行檔是否在系統路徑上。例如，如果 Python 3 可用，請使用 onpath python3略過此生命週期步驟。
- exists *file* – 檢查檔案是否存在。例如，如果 /tmp/my-configuration.db 存在，請使用 exists /tmp/my-configuration.db略過此生命週期步驟。

Timeout

(選用) 在 AWS IoT Greengrass Core 軟體終止程序之前，指令碼可執行的時間上限，以秒為單位。

預設：120 秒

Setenv

(選用) 要提供給指令碼的環境變數字典。這些環境變數會覆寫您在 中提供的變數Lifecycle.Setenv。

run

(選用) 物件或字串，定義元件啟動時要執行的指令碼。

此生命週期步驟執行時，元件會進入 RUNNING 狀態。如果run指令碼以成功碼結束，元件會進入 STOPPING 狀態。如果指定shutdown指令碼，則會執行；否則元件會進入 FINISHED 狀態。

相依於此元件的元件會在此生命週期步驟執行時啟動。若要執行背景程序，例如相依元件使用的服務，請改用startup生命週期步驟。

當您使用run生命週期部署元件時，核心裝置可以在此生命週期指令碼執行後立即報告部署。因此，即使run生命週期指令碼在執行後很快失敗，部署也可以完成且成功。如果您希望部署狀態取決於元件啟動指令碼的結果，請改用startup生命週期步驟。

Note

您只能定義一個 startup或 run生命週期。

此物件或字串包含下列資訊：

Script

要執行的指令碼。

RequiresPrivilege

(選用) 您可以使用根權限執行指令碼。如果您將此選項設定為 true，則 AWS IoT Greengrass Core 軟體會執行此生命週期指令碼做為根目錄，而不是您設定為執行此元件的系統使用者。預設為 false。

Skipif

(選用) 檢查以判斷是否執行指令碼。您可以定義 `skipif` 來檢查可執行檔是否在路徑上，或檔案是否存在。如果輸出為 `true`，則 AWS IoT Greengrass Core 軟體會略過 步驟。選擇下列其中一個檢查：

- `onpath runnable` – 檢查可執行檔是否在系統路徑上。例如，如果 Python 3 可用，請使用 `onpath python3` 略過此生命週期步驟。
- `exists file` – 檢查檔案是否存在。例如，如果 `/tmp/my-configuration.db` 存在，請使用 `exists /tmp/my-configuration.db` 略過此生命週期步驟。

Timeout

(選用) 在 AWS IoT Greengrass Core 軟體終止程序之前，指令碼可執行的時間上限，以秒為單位。

根據預設，此生命週期步驟不會逾時。如果您省略此逾時，`run` 指令碼會執行直到結束為止。

Setenv

(選用) 要提供給指令碼的環境變數字典。這些環境變數會覆寫您在 `configuration.json` 中提供的變數 `Lifecycle.Setenv`。

startup

(選用) 物件或字串，定義元件啟動時要執行的背景程序。

使用 `startup` 執行命令，該命令必須成功結束，或將元件的狀態更新為 `RUNNING`，然後相依元件才能啟動。使用 [UpdateState](#) IPC 操作，將元件的狀態設定為 `RUNNING` 或當元件啟動未結束的指令碼 `ERRORED` 時。例如，您可以定義使用 `startup` 的步驟 `etc/init.d/mysqld start`。

此生命週期步驟執行時，元件會進入 `STARTING` 狀態。如果 `startup` 指令碼以成功碼結束，元件會進入 `RUNNING` 狀態。然後，相依元件可以啟動。

當您使用 `startup` 生命週期部署元件時，核心裝置可以在此生命週期指令碼結束或報告其狀態後，將部署報告為完成。換句話說，部署的狀態是直到所有元件的啟動指令碼結束或報告狀態 `IN_PROGRESS` 為止。

Note

您只能定義一個 `startup` 或 `run` 生命週期。

此物件或字串包含下列資訊：

Script

要執行的指令碼。

RequiresPrivilege

(選用) 您可以使用根權限執行指令碼。如果您將此選項設定為 `true`，則 AWS IoT Greengrass 核心軟體會執行此生命週期指令碼做為根目錄，而不是您設定為執行此元件的系統使用者。預設為 `false`。

Skipif

(選用) 檢查以判斷是否執行指令碼。您可以定義 來檢查可執行檔是否在路徑上，或檔案是否存在。如果輸出為 `true`，則 AWS IoT Greengrass Core 軟體會略過 步驟。選擇下列其中一項檢查：

- `onpath runnable` – 檢查可執行檔是否在系統路徑上。例如，如果 Python 3 可用，請使用 `onpath python3`略過此生命週期步驟。
- `exists file` – 檢查檔案是否存在。例如，如果 `/tmp/my-configuration.db` 存在，請使用 `exists /tmp/my-configuration.db`略過此生命週期步驟。

Timeout

(選用) 在 AWS IoT Greengrass Core 軟體終止程序之前，指令碼可執行的時間上限，以秒為單位。

預設：120 秒

Setenv

(選用) 要提供給指令碼的環境變數字典。這些環境變數會覆寫您在 中提供的變數 `Lifecycle.Setenv`。

shutdown

(選用) 物件或字串，定義元件關閉時要執行的指令碼。使用關閉生命週期來執行您希望在元件處於 STOPPING 狀態時執行的程式碼。關機生命週期可用來停止由 `startup`或 `run`指令碼啟動的程序。

如果您在 中啟動背景程序 `startup`，請使用 `shutdown` 步驟在元件關閉時停止該程序。例如，您可以定義使用 停止 MySQL 程序 `shutdown`的步驟 `/etc/init.d/mysqld stop`。

`shutdown` 指令碼會在元件進入 STOPPING 狀態後執行。如果指令碼成功完成，元件會進入 FINISHED 狀態。

此物件或字串包含下列資訊：

Script

要執行的指令碼。

RequiresPrivilege

(選用) 您可以使用根權限執行指令碼。如果您將此選項設定為 `true`，則 AWS IoT Greengrass 核心軟體會執行此生命週期指令碼做為根目錄，而不是您設定為執行此元件的系統使用者。預設為 `false`。

Skipif

(選用) 檢查以判斷是否執行指令碼。您可以定義 來檢查可執行檔是否在路徑上，或檔案是否存在。如果輸出為 `true`，則 AWS IoT Greengrass Core 軟體會略過 步驟。選擇下列其中一項檢查：

- `onpath` **runnable** – 檢查可執行檔是否在系統路徑上。例如，如果 Python 3 可用，請使用 `onpath python3`略過此生命週期步驟。
- `exists` **file** – 檢查檔案是否存在。例如，如果 `/tmp/my-configuration.db` 存在，請使用 `exists /tmp/my-configuration.db`略過此生命週期步驟。

Timeout

(選用) 在 AWS IoT Greengrass Core 軟體終止程序之前，指令碼可執行的時間上限，以秒為單位。

預設：15 秒。

Setenv

(選用) 要提供給指令碼的環境變數字典。這些環境變數會覆寫您在 中提供的變數 `Lifecycle.Setenv`。

recover

(選用) 物件或字串，定義元件發生錯誤時要執行的指令碼。

此步驟會在元件進入 `ERROR` 狀態時執行。如果元件變成 `ERROR` 三次但未成功復原，則元件會變更為 `BROKEN` 狀態。若要修正 `BROKEN` 元件，您必須再次部署。

此物件或字串包含下列資訊：

Script

要執行的指令碼。

RequiresPrivilege

(選用) 您可以使用根權限執行指令碼。如果您將此選項設定為 `true`，則 AWS IoT Greengrass 核心軟體會執行此生命週期指令碼做為根目錄，而不是您設定為執行此元件的系統使用者。預設為 `false`。

Skipif

(選用) 檢查以判斷是否執行指令碼。您可以定義 來檢查可執行檔是否在路徑上，或檔案是否存在。如果輸出為 `true`，則 AWS IoT Greengrass Core 軟體會略過 步驟。選擇下列其中一項檢查：

- `onpath` *runnable* – 檢查可執行檔是否在系統路徑上。例如，如果 Python 3 可用，請使用 `onpath python3` 略過此生命週期步驟。
- `exists` *file* – 檢查檔案是否存在。例如，如果 `/tmp/my-configuration.db` 存在，請使用 `exists /tmp/my-configuration.db` 略過此生命週期步驟。

Timeout

(選用) 在 AWS IoT Greengrass Core 軟體終止程序之前，指令碼可執行的時間上限，以秒為單位。

預設：60 秒。

Setenv

(選用) 要提供給指令碼的環境變數字典。這些環境變數會覆寫您在 中提供的變數 `Lifecycle.Setenv`。

bootstrap

(選用) 物件或字串，定義需要 AWS IoT Greengrass Core 軟體或核心裝置重新啟動的指令碼。這可讓您開發元件，在安裝作業系統更新或執行時間更新後執行重新啟動。

Note

若要安裝不需要 AWS IoT Greengrass Core 軟體或裝置重新啟動的更新或相依性，請使用 [安裝生命週期](#)。

在 AWS IoT Greengrass Core 軟體部署元件的下列情況下，此生命週期步驟會在安裝生命週期步驟之前執行：

- 元件第一次部署到核心裝置。
- 元件版本會變更。
- 引導指令碼會隨著元件組態更新而變更。

AWS IoT Greengrass 核心軟體完成部署中具有引導步驟之所有元件的引導步驟後，軟體會重新啟動。

 Important

您必須將 AWS IoT Greengrass Core 軟體設定為系統服務，才能重新啟動 AWS IoT Greengrass Core 軟體或核心裝置。如果您未將 AWS IoT Greengrass Core 軟體設定為系統服務，則軟體不會重新啟動。如需詳細資訊，請參閱[將 Greengrass 核設定為系統服務](#)。

此物件或字串包含下列資訊：

BootstrapOnRollback

 Note

啟用此功能時，BootstrapOnRollback 只會針對已完成或嘗試執行引導生命週期步驟的元件執行，做為失敗目標部署的一部分。此功能適用於 Greengrass 核版本 2.12.0 和更新版本。

(選用) 您可以在轉返部署中執行引導生命週期步驟。如果您將此選項設定為 `true`，則會執行復原部署中定義的引導生命週期步驟。部署失敗時，元件引導生命週期的先前版本會在復原部署期間再次執行。

預設為 `false`。

Script

要執行的指令碼。此指令碼的結束程式碼定義重新啟動指示。使用下列結束代碼：

- 0 – 請勿重新啟動 AWS IoT Greengrass Core 軟體或核心裝置。AWS IoT Greengrass 核心軟體仍會在所有元件啟動後重新啟動。
- 100 – 請求重新啟動 AWS IoT Greengrass Core 軟體。

- 101 – 請求重新啟動核心裝置。

出口代碼 100 到 199 保留給特殊行為。其他結束代碼代表指令碼錯誤。

RequiresPrivilege

(選用) 您可以使用根權限執行指令碼。如果您將此選項設定為 `true`，則 AWS IoT Greengrass 核心軟體會執行此生命週期指令碼做為根目錄，而不是您設定為執行此元件的系統使用者。預設為 `false`。

Timeout

(選用) 在 AWS IoT Greengrass Core 軟體終止程序之前，指令碼可執行的時間上限，以秒為單位。

預設：120 秒

Setenv

(選用) 要提供給指令碼的環境變數字典。這些環境變數會覆寫您在 `Lifecycle` 中提供的變數 `Lifecycle.Setenv`。

Selections

(選用) 選取索引鍵清單，指定要為此資訊清單執行的 [全域生命週期](#) 區段。在全域生命週期中，您可以使用任何層級的選擇索引鍵來定義生命週期步驟，以選取生命週期的子區段。然後，核心裝置會使用符合此資訊清單中選取索引鍵的這些區段。如需詳細資訊，請參閱 [全域生命週期範例](#)。

Important

只有在此資訊清單未定義生命週期時，核心裝置才會使用全域生命週期中的選擇。

您可以指定 `all` 選取金鑰，以執行沒有選取金鑰的全域生命週期區段。

Artifacts

(選用) 物件清單，每個物件都會為此資訊清單定義的平台上的元件定義二進位成品。例如，您可以將程式碼或影像定義為成品。

當元件部署時，AWS IoT Greengrass 核心軟體會將成品下載到核心裝置上的資料夾。您也可以將成品定義為軟體下載後擷取的封存檔案。

您可以使用[配方變數](#)來取得成品安裝在核心裝置上的資料夾路徑。

- 一般檔案 – 使用 [artifacts : path 配方變數](#)來取得包含成品之資料夾的路徑。例如，在配方`{artifacts:path}/my_script.py`中指定，以取得具有 URI 的成品路徑`s3://amzn-s3-demo-bucket/path/to/my_script.py`。
- 擷取的封存 – 使用 [artifacts : decompressedPath 配方變數](#)來取得包含擷取的封存成品之資料夾的路徑。Core AWS IoT Greengrass 軟體會將每個封存擷取到與封存同名的資料夾。例如，在配方`{artifacts:decompressedPath}/my_archive/my_script.py`中指定，以在具有 URI 的封存成品`my_script.py`中取得的路徑`s3://amzn-s3-demo-bucket/path/to/my_archive.zip`。

 Note

當您在本機核心裝置上開發具有封存成品的元件時，您可能沒有該成品的 URI。若要使用擷取成品Unarchive的選項測試元件，請指定檔案名稱符合封存成品檔案名稱的 URI。您可以指定預期上傳封存成品的 URI，也可以指定新的預留位置 URI。例如，若要在本機部署期間擷取`my_archive.zip`成品，您可以指定 `s3://amzn-s3-demo-bucket/my_archive.zip`。

每個物件都包含下列資訊：

Uri

S3 儲存貯體中成品的 URI。AWS IoT Greengrass 核心軟體會在元件安裝時從此 URI 擷取成品，除非成品已存在於裝置上。每個成品在每個資訊清單中都必須具有唯一的檔案名稱。

Unarchive

(選用) 要解壓縮的封存類型。您可以從以下選項中選擇：

- NONE – 檔案不是要解壓縮的封存。AWS IoT Greengrass 核心軟體會將成品安裝到核心裝置上的資料夾。您可以使用 [artifacts : path 配方變數](#)來取得此資料夾的路徑。
- ZIP – 檔案是 ZIP 封存檔。Core AWS IoT Greengrass 軟體會將封存擷取到與封存同名的資料夾。您可以使用 [artifacts : decompressedPath 配方變數](#)，取得包含此資料夾的資料夾路徑。

預設為 NONE。

Permission

(選用) 定義要為此成品檔案設定的存取許可的物件。您可以設定讀取許可和執行許可。

 Note

您無法設定寫入許可，因為 AWS IoT Greengrass Core 軟體不允許元件編輯成品資料夾中的成品檔案。若要編輯元件中的成品檔案，請將其複製到另一個位置，或發佈和部署新的成品檔案。

如果您將成品定義為要解壓縮的封存，則 AWS IoT Greengrass Core 軟體會針對從封存解壓縮的檔案設定這些存取許可。Core AWS IoT Greengrass 軟體會將資料夾的存取許可設定為 ALL 適用於 Read 和的 Execute。這可讓元件檢視 資料夾中的解壓縮檔案。若要設定封存中個別檔案的許可，您可以在[安裝生命週期指令碼](#)中設定許可。

此物件包含下列資訊：

Read

(選用) 為此成品檔案設定的讀取許可。若要允許其他元件存取此成品，例如相依於此元件的元件，請指定 ALL。您可以從以下選項中選擇：

- NONE – 檔案無法讀取。
- OWNER – 您設定為執行此元件的系統使用者可讀取該檔案。
- ALL – 所有使用者都可讀取檔案。

預設為 OWNER。

Execute

(選用) 為此成品檔案設定的執行許可。Execute 許可表示 Read 許可。例如，如果您 ALL 為 指定 Execute，則所有使用者都可以讀取和執行此成品檔案。

您可以從以下選項中選擇：

- NONE – 檔案無法執行。
- OWNER – 檔案可由您設定為執行元件的系統使用者執行。
- ALL – 所有使用者皆可執行 檔案。

預設為 NONE。

Digest

(唯讀) 成品的密碼編譯摘要雜湊。當您建立元件時，AWS IoT Greengrass 會使用雜湊演算法來計算成品檔案的雜湊。然後，當您部署元件時，Greengrass 核會計算下載成品的雜

湊，並將雜湊與此摘要進行比較，以在安裝之前驗證成品。如果雜湊不符合摘要，則部署會失敗。

如果您設定此參數，會 AWS IoT Greengrass 取代您在建立元件時設定的值。

Algorithm

(唯讀) AWS IoT Greengrass 用來計算成品摘要雜湊的雜湊演算法。

如果您設定此參數，會 AWS IoT Greengrass 取代您在建立元件時設定的值。

Lifecycle

定義如何安裝和執行元件的物件。只有在要使用的資訊清單未指定生命週期時，核心裝置才會使用全域生命週期。

Note

您可以在資訊清單之外定義此生命週期。您也可以定義資訊清單生命週期，套用至符合該資訊清單的平台。

在全域生命週期中，您可以指定針對每個資訊清單中指定的特定選擇索引鍵執行的生命週期。選擇索引鍵是字串，可識別要針對每個資訊清單執行的全域生命週期區段。

all 選擇索引鍵是任何沒有選擇索引鍵區段的預設值。這表示您可以在資訊清單中指定all選擇金鑰，以執行全域生命週期的區段，而無需選擇金鑰。您不需要在全域生命週期中指定all選擇索引鍵。

如果資訊清單未定義生命週期或選擇項目索引鍵，核心裝置會預設為使用all選擇項目。這表示在此情況下，核心裝置會使用未使用選擇金鑰的全域生命週期區段。

此物件包含的資訊與資訊清單生命週期相同，但您可以在任何層級指定選擇索引鍵，以選擇生命週期的子區段。

Tip

我們建議您針對每個選擇索引鍵僅使用小寫字母，以避免選擇索引鍵與生命週期索引鍵之間發生衝突。生命週期金鑰以大寫字母開頭。

Example具有最上層選取索引鍵的全域生命週期範例

```
Lifecycle:
  key1:
    install:
      SkipIf: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Example具有底層選取索引鍵的全域生命週期範例

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Example具有多個選擇索引鍵層級的全域生命週期範例

```
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
```

配方變數

配方變數會公開目前元件和核心中的資訊，供您在配方中使用。例如，您可以使用配方變數，將元件組態參數傳遞至您在生命週期指令碼中執行的應用程式。

您可以在元件配方的下列區段中使用配方變數：

- 生命週期定義。
- 元件組態定義，如果您使用 [Greengrass 核 v2.6.0](#) 或更新版本，並將 [interpolateComponentConfiguration](#) 組態選項設定為 true。您也可以使用 [部署元件組態更新](#) 時使用配方變數。

配方變數使用 {recipe_variable} 語法。大括號表示配方變數。

AWS IoT Greengrass 支援下列配方變數：

component_dependency_name:configuration:json_pointer

此配方定義之元件的組態參數值，或此元件所依賴之元件的組態參數值。

您可以使用此變數，為您在元件生命週期中執行的指令碼提供參數。

Note

AWS IoT Greengrass 僅在元件生命週期定義中支援此配方變數。

此配方變數具有下列輸入：

- component_dependency_name – (選用) 要查詢的元件相依性名稱。省略此區段以查詢此配方定義的元件。您只能指定直接相依性。
- json_pointer – 要評估之組態值的 JSON 指標。JSON 指標以正斜線 開頭 /。若要識別巢狀元件組態中的值，請使用斜線 (/) 來分隔組態中每個層級的索引鍵。您可以使用數字做為索引鍵，在清單中指定索引。如需詳細資訊，請參閱 [JSON 指標規格](#)。

AWS IoT Greengrass Core 對 YAML 格式的配方使用 JSON 指標。

JSON 指標可以參考下列節點類型：

- value node. AWS IoT Greengrass Core 會將配方變數取代為值的字串表示法。Null 值 null 會轉換為字串。

- 物件節點。AWS IoT Greengrass Core 會將配方變數取代為該物件的序列化 JSON 字串表示法。
- 無 node。AWS IoT Greengrass Core 不會取代配方變數。

例如，{configuration:/Message} 配方變數會擷取元件組態中 Message 金鑰的值。{com.example.MyComponentDependency:configuration:/server/port} 配方變數會在元件相依性的 server 組態物件 port 中擷取 的值。

component_dependency_name:artifacts:path

此配方定義元件的成品根路徑，或此元件所依賴的元件的成品根路徑。

當元件安裝時，會將元件的成品 AWS IoT Greengrass 複製到此變數公開的資料夾。您可以使用此變數來識別要在元件生命週期中執行的指令碼位置，例如。

此路徑的資料夾為唯讀。若要修改成品檔案，請將檔案複製到另一個位置，例如目前的工作目錄 (\$PWD 或 .)。然後，修改其中的檔案。

若要從元件相依性讀取或執行成品，該成品的 Read 或 Execute 許可必須是 ALL。如需詳細資訊，請參閱您在元件配方中定義的[成品許可](#)。

此配方變數具有下列輸入：

- component_dependency_name – (選用) 要查詢的元件相依性名稱。省略此區段以查詢此配方定義的元件。您只能指定直接相依性。

component_dependency_name:artifacts:decompressedPath

此配方定義之元件的解壓縮封存成品根路徑，或此元件所依賴之元件的解壓縮封存成品根路徑。

當元件安裝時，會將元件的封存成品 AWS IoT Greengrass 解壓縮至此變數公開的資料夾。您可以使用此變數來識別要在元件生命週期中執行的指令碼位置，例如。

每個成品都會解壓縮到解壓縮路徑中的資料夾，其中資料夾的名稱與成品的名稱相同，減去其副檔名。例如，名為 models.zip 的 ZIP 成品會解壓縮至 {artifacts:decompressedPath}/models 資料夾。

此路徑的資料夾為唯讀。若要修改成品檔案，請將檔案複製到另一個位置，例如目前的工作目錄 (\$PWD 或 .)。然後，修改其中的檔案。

若要從元件相依性讀取或執行成品，該成品的 Read 或 Execute 許可必須是 ALL。如需詳細資訊，請參閱您在元件配方中定義的[成品許可](#)。

此配方變數具有下列輸入：

- `component_dependency_name` – (選用) 要查詢的元件相依性名稱。省略此區段以查詢此配方定義的元件。您只能指定直接相依性。

`component_dependency_name`:work:path

此功能適用於 [Greengrass 核元件](#) 的 v2.0.4 和更新版本。

此配方定義之元件的工作路徑，或此元件所依賴之元件的工作路徑。此配方變數的值等同於從元件內容執行時 `$PWD` 環境變數和 `pwd` 命令的輸出。

您可以使用此配方變數，在元件與相依性之間共用檔案。

此路徑的資料夾可由此配方定義的元件以及以相同使用者和群組身分執行的其他元件讀取和寫入。

此配方變數具有下列輸入：

- `component_dependency_name` – (選用) 要查詢的元件相依性名稱。省略此區段以查詢此配方定義的元件。您只能指定直接相依性。

`kernel:rootPath`

AWS IoT Greengrass 核心根路徑。

`iot:thingName`

此功能適用於 [Greengrass 核元件](#) 的 v2.3.0 和更新版本。

核心裝置 AWS IoT 物件的名稱。

配方範例

您可以參考下列配方範例，協助您建立元件的配方。

AWS IoT Greengrass 策劃 Greengrass 元件的索引，稱為 Greengrass 軟體目錄。此目錄會追蹤由 Greengrass 社群開發的 Greengrass 元件。您可以從此目錄中下載、修改和部署元件，以建立 Greengrass 應用程式。如需詳細資訊，請參閱[社群元件](#)。

主題

- [Hello World 元件配方](#)
- [Python 執行期元件範例](#)
- [指定數個欄位的元件配方](#)

Hello World 元件配方

下列配方說明執行 Python 指令碼的 Hello World 元件。此元件支援所有平台，並接受將 Message AWS IoT Greengrass 參數做為引數傳遞至 Python 指令碼。這是[入門教學](#)課程中 Hello World 元件的配方。

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}"

```

Python 執行期元件範例

下列配方說明安裝 Python 的元件。此元件支援 64 位元 Linux 裝置。

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
```

指定數個欄位的元件配方

下列元件配方使用數個配方欄位。

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'
```

元件環境變數參考

AWS IoT Greengrass 核心軟體會在執行元件的生命週期指令碼時設定環境變數。您可以在元件中取得這些環境變數，以取得物件名稱 AWS 區域和 Greengrass 核版本。軟體也會設定您的元件使用[程序間通訊 SDK](#) 和 [與 AWS 服務互動](#)所需的環境變數。

您也可以為元件的生命週期指令碼設定自訂環境變數。如需詳細資訊，請參閱 [Setenv](#)。

Core AWS IoT Greengrass 軟體會設定下列環境變數：

`AWS_IOT_THING_NAME`

代表此 Greengrass 核心裝置的 AWS IoT 物件名稱。

`AWS_REGION`

此 Greengrass 核心裝置運作 AWS 區域 所在的 。

AWS SDKs 使用此環境變數來識別要使用的預設區域。此變數等同於 `AWS_DEFAULT_REGION`。

`AWS_DEFAULT_REGION`

此 Greengrass 核心裝置運作 AWS 區域 所在的 。

AWS CLI 使用此環境變數來識別要使用的預設區域。此變數等同於 `AWS_REGION`。

`GGC_VERSION`

在此 [Greengrass 核心裝置上執行的 Greengrass 核元件](#)版本。

`GG_ROOT_CA_PATH`

此功能適用於 [Greengrass 核元件](#)的 v2.5.5 和更新版本。

Greengrass 核使用的根憑證授權機構 (CA) 憑證路徑。

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

元件用來與 AWS IoT Greengrass Core 軟體通訊的 IPC 通訊端路徑。如需詳細資訊，請參閱[使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#)。

`SVCUID`

元件用來連線至 IPC 通訊端並與 AWS IoT Greengrass Core 軟體通訊的秘密字符。如需詳細資訊，請參閱[使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#)。

AWS_CONTAINER_AUTHORIZATION_TOKEN

元件用來從字符[交換服務元件擷取憑證的秘密字符](#)。

AWS_CONTAINER_CREDENTIALS_FULL_URI

元件請求從[字符交換服務元件](#)擷取憑證的 URI。

將 AWS IoT Greengrass 元件部署至裝置

您可以使用 AWS IoT Greengrass 將元件部署至裝置或裝置群組。您可以使用部署來定義傳送至裝置的元件和組態。會 AWS IoT Greengrass 部署到代表 Greengrass 核心裝置的目標、AWS IoT 物件或物件群組。AWS IoT Greengrass 會使用[AWS IoT Core 任務](#)來部署到您的核心裝置。您可以設定任務如何推展至您的裝置。

核心裝置部署

每個核心裝置都會執行該裝置部署的元件。相同目標的新部署會覆寫目標的先前部署。建立部署時，您可以定義要套用至核心裝置現有軟體的元件和組態。

當您修改目標的部署時，您可以將先前修訂版中的元件取代為新修訂版中的元件。例如，您將 [日誌管理員](#)和 [秘密管理員](#)元件部署到物件群組 TestGroup。然後，您可以為 建立另一個部署TestGroup，只指定秘密管理員元件。因此，該群組中的核心裝置不再執行日誌管理員。

平台相依性解析

當核心裝置收到部署時，它會檢查以確保元件與核心裝置相容。例如，如果您將 部署[Firehose](#)到 Windows 目標，部署將會失敗。

元件相依性解析

在元件部署期間，核心裝置會驗證物件群組中所有元件相依性和版本需求的相容性。此驗證可確保在繼續部署之前，所有元件及其相依性的版本限制都受到滿足。

相依性解析程序從識別其配方中沒有相依性的目標元件開始。然後，系統會使用廣度優先搜尋 (BFS) 建構相依性樹狀結構，系統性地探索每個目標節點，並在繼續下一個節點之前先尋找其相依性。每個節點都包含目標元件做為 金鑰，並將版本需求做為 值。

版本需求結合三組限制條件：

- 已在現有物件群組中建立的版本需求。
- 部署所需的元件版本。您必須在進行或更新部署時選取元件版本。
- 在配方相依性區段中定義的任何元件版本限制。

解決元件相依性

在部署期間，Greengrass 核會先嘗試尋找目前在滿足需求的裝置上執行的本機候選項目。如果執行中的元件滿足需求，則核會從配方資料夾取得儲存的配方路徑，並在本機存放區中找到最新的本機版本。

對於 AWS 雲端 部署，核接著會呼叫 [ResolveComponentCandidates API](#)。此 API 將從最新的可用版本開始，並檢查是否符合相依性和要求。當核心從 API 取得回應時，它會選取該最新版本。如果找不到符合 AWS 雲端 需求的 版本，則部署會失敗。如果裝置離線，則會回到找到的原始本機候選項目。如果找不到符合需求的本機候選項目，則部署會失敗。

對於本機部署，如果存在且符合要求，則核只會使用本機候選項目，而不會進行協商 AWS 雲端。如果沒有這類候選項目，則部署會失敗。

Note

所有已解析的配方都會儲存在本機以供日後參考。

如需詳細資訊，請參閱 GitHub 中的[相依性解析](#)一節。

如果 Greengrass 核能夠成功解析所有元件，則核日誌將包含以下行。

```
resolve-all-group-dependencies-finish. Finish resolving all groups dependencies.
```

Example

以下是 核如何解決元件需求的範例。

- 您可以部署ComponentAComponentC。
- 您也可以部署取決於 ComponentC 1.4.0-1.9.5 版的 ComponentBComponentB 。

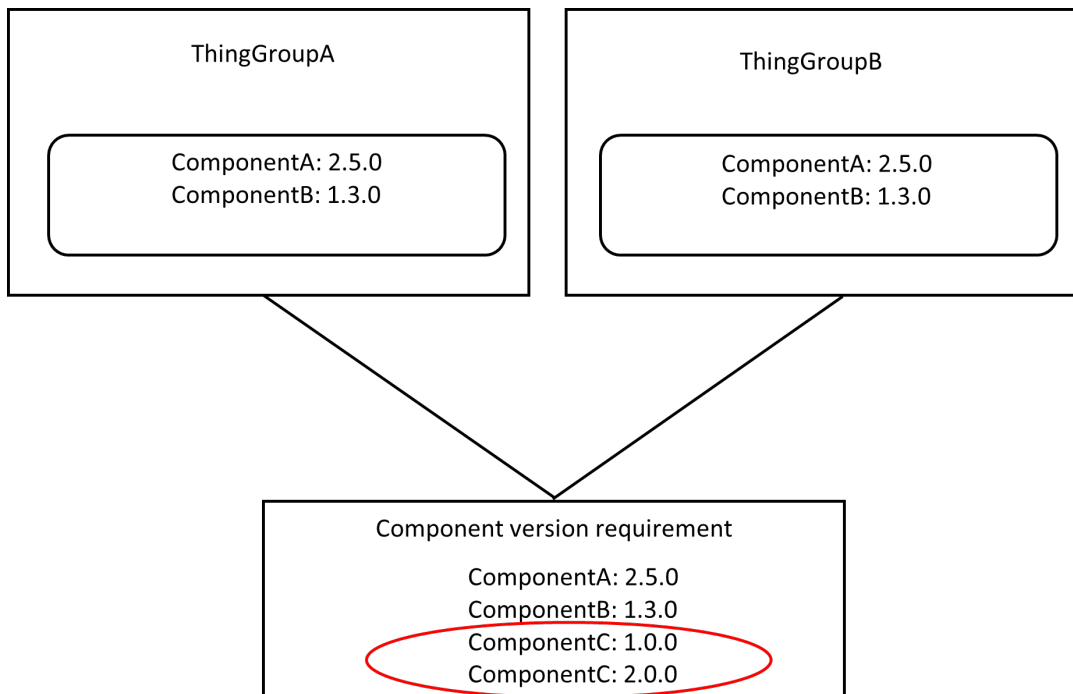
透過元件相依性解析，核會部署最新版本的 ComponentC 版本，以滿足 ComponentA 和 ComponentB 的需求。此最新版本的 ComponentC 是 1.9.0 版。

常見元件相依性解析失敗

元件相依性解析可能會因為兩個主要原因失敗：目標版本需求衝突或元件相依性需求衝突。

案例 1：目標版本需求衝突

- 物件存在於一個物件群組中，您也想要將該物件新增至新的物件群組。如果新物件群組需要不同的物件版本，則部署會失敗。
- 如果物件屬於物件群組，且想要透過物件部署更新元件版本，則部署也可能會失敗。



失敗日誌範例：

```
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(ComponentManager
    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

```

日誌指出版本衝突錯誤，因為核心找不到同時符合兩個衝突需求的元件版本。

如何解決此問題

- 如果您想要在每個物件群組中保留元件，請在每個物件群組中選取該元件的相同版本。
- 選取符合部署需求的元件版本。

- 如果您想要使用不符合物件群組需求的元件版本，請選取符合物件群組版本需求的元件版本，並僅在該物件群組中使用該元件。

案例 2：元件相依性版本需求衝突

如果元件是不同元件的相依性，且元件需要不同版本或該元件的不同版本範圍，則可能沒有可用的版本可滿足所有版本需求。在此案例中，部署將會失敗。

Example

ComponentA (v2.5.0)、ComponentB (v1.3.0) 和 ComponentC (v1.0.0) 的部署

- ComponentA 要求 ComponentB 版本 $\geq 1.0.0$ 。

```
---
...
ComponentName: ComponentA
ComponentVersion: "2.5.0"
ComponentDependencies:
  ComponentB:
    VersionRequirement: " $\geq 1.0.0$ "
    DependencyType: "HARD"
...
```

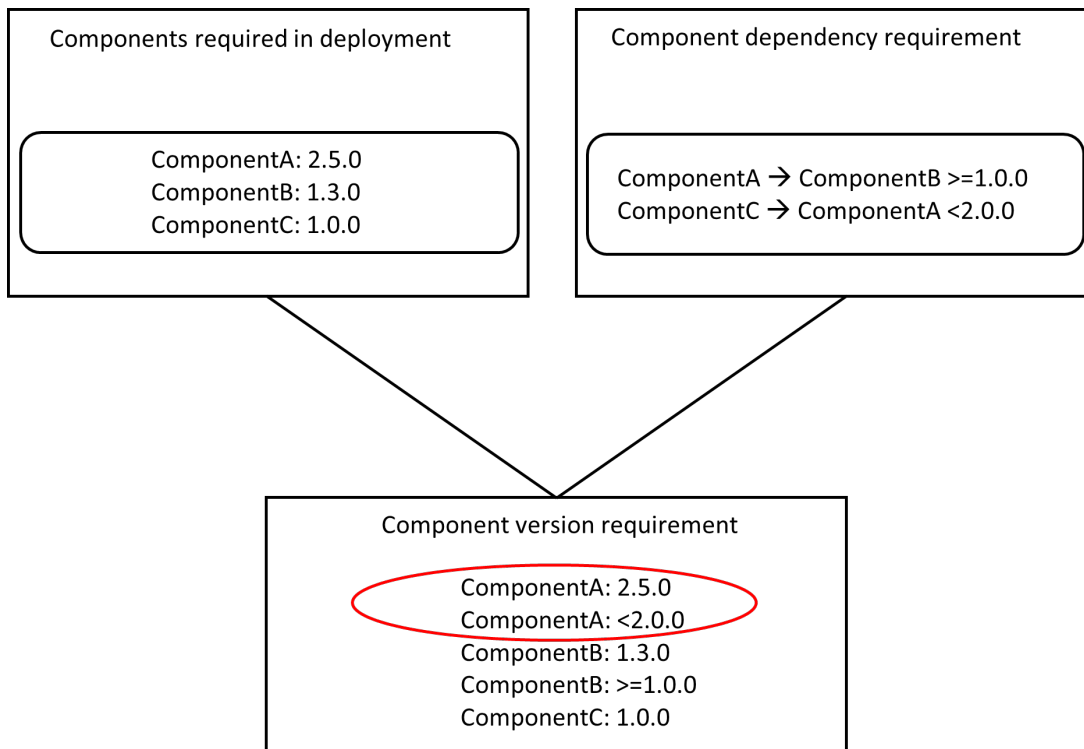
- ComponentC 需要 ComponentA 版本 $< 2.0.0$ 。

```
---
...
ComponentName: ComponentC
ComponentVersion: "1.0.0"
ComponentDependencies:
  ComponentA:
    VersionRequirement: " $< 2.0.0$ "
    DependencyType: "HARD"
...
```

ComponentA 的兩個要求之間存在版本衝突：

- 此部署中 ComponentA 需要 2.5.0 版
- ComponentC 需要低於 2.0.0 的 ComponentA 版本

這兩個要求彼此矛盾，使得核心無法找到滿足這兩個要求的 ComponentA 版本。因此，相依性解析會失敗。



失敗日誌範例：

```
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(ComponentManager.
    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

```

日誌指出 核找不到符合下列需求的 ComponentA 版本。

- ComponentA 的要求完全是 2.5.0 版（來自 ThingGroupA）。
- 使用低於 2.0.0 的 ComponentC 版本的需求。

如何解決此問題

- 如果您想要在每個物件群組中保留元件，請在每個物件群組中選取該元件的相同版本。
- 選取符合部署需求的元件版本。
- 如果您想要使用不符合物件群組需求的元件版本，請選取符合物件群組版本需求的元件版本，並僅在該物件群組中使用該元件。

 Tip

如果您在任何 AWS 提供的元件上看到此錯誤，您可以透過將衝突的元件更新到最新版本來解決它。

從物件群組移除裝置

當您從物件群組移除核心裝置時，元件部署行為取決於核心裝置執行的 [Greengrass 核心](#) 版本。

2.5.1 and later

當您從物件群組移除核心裝置時，行為取決於 AWS IoT 政策是否授予 `greengrass:ListThingGroupsForCoreDevice` 許可。如需核心裝置此許可和 AWS IoT 政策的詳細資訊，請參閱 [的裝置身分驗證和授權 AWS IoT Greengrass](#)。

- 如果 AWS IoT 政策授予此許可

當您從物件群組移除核心裝置時，下次對裝置進行部署時，會 AWS IoT Greengrass 移除物件群組的元件。如果裝置的元件包含在下一次部署中，則該元件不會從裝置中移除。

- 如果 AWS IoT 政策未授予此許可

當您從物件群組中移除核心裝置時，AWS IoT Greengrass 不會從裝置刪除該物件群組的元件。

若要從裝置移除元件，請使用 Greengrass CLI 的 [部署建立](#) 命令。使用 `--remove` 引數指定要移除的元件，並使用 `--groupId` 引數指定物件群組。

2.5.0

當您從物件群組移除核心裝置時，下次對裝置進行部署時，會 AWS IoT Greengrass 移除物件群組的元件。如果裝置的元件包含在下一次部署中，則該元件不會從裝置中移除。

此行為需要核心裝置的 AWS IoT 政策授予 `greengrass:ListThingGroupsForCoreDevice` 許可。如果核心裝置沒有此許可，則核心裝置無法套用部署。如需詳細資訊，請參閱 [的裝置身分驗證和授權 AWS IoT Greengrass](#)。

2.0.x - 2.4.x

當您從物件群組中移除核心裝置時，AWS IoT Greengrass 不會從裝置刪除該物件群組的元件。

若要從裝置移除元件，請使用 Greengrass CLI 的[部署建立](#)命令。使用 `--remove` 引數指定要移除的元件，並使用 `--groupId` 引數指定物件群組。

部署

部署是連續的。當您建立部署時，會將部署 AWS IoT Greengrass 轉出至線上的目標裝置。如果目標裝置未上線，則下次連線時會接收部署 AWS IoT Greengrass。當您將核心裝置新增至目標物件群組時，會將該物件群組的最新部署 AWS IoT Greengrass 傳送給裝置。

在核心裝置部署元件之前，預設會通知裝置上的每個元件。Greengrass 元件可以回應通知以延遲部署。如果裝置電池電量低或正在執行無法中斷的程序，建議您延遲部署。如需詳細資訊，請參閱[教學課程：開發可延遲元件更新的 Greengrass 元件](#)。建立部署時，您可以將其設定為部署，而無需通知元件。

每個目標物件或物件群組一次可以有一個部署。這表示當您為目標建立部署時，AWS IoT Greengrass 不會再部署該目標部署的先前修訂版。

部署選項

部署提供多種選項，可讓您控制哪些裝置接收更新，以及更新如何部署。建立部署時，您可以設定下列選項：

- AWS IoT Greengrass 元件

定義要在目標 device. AWS IoT Greengrass components 上安裝和執行的元件是您在 Greengrass 核心裝置上部署和執行的軟體模組。只有在元件支援裝置的平台時，裝置才會接收元件。這可讓您部署至裝置群組，即使目標裝置在多個平台上執行也一樣。如果元件不支援裝置的平台，則元件不會部署到裝置。

您可以將自訂元件和 AWS 提供的元件部署至您的裝置。當您部署元件時，會 AWS IoT Greengrass 識別任何元件相依性，也會進行部署。如需詳細資訊，請參閱[開發 AWS IoT Greengrass 元件及 AWS 提供的元件](#)。

您可以定義要為每個元件部署的版本和組態更新。組態更新指定如何在核心裝置上修改元件的現有組態，如果元件不存在於核心裝置上，則修改元件的預設組態。您可以指定要重設為預設值的組態值，以及要合併到核心裝置的新組態值。當核心裝置收到不同目標的部署，且每個部署指定相容的元件版本時，核心裝置會根據您建立部署時的時間戳記，依序套用組態更新。如需詳細資訊，請參閱[更新元件組態](#)。

⚠ Important

部署元件時，會 AWS IoT Greengrass 安裝該元件所有相依性的最新支援版本。因此，如果您將新裝置新增至物件群組，或者您更新以這些裝置為目標的部署，則可能會自動將 AWS 新修補程式版本的公有元件部署到您的核心裝置。有些自動更新，例如核更新，可能會導致您的裝置意外重新啟動。

為了防止在您裝置上執行之元件的意外更新，我們建議您在[建立部署](#)時直接包含該元件的偏好版本。如需 AWS IoT Greengrass Core 軟體更新行為的詳細資訊，請參閱[更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

- 部署政策

定義何時可以安全地部署組態，以及部署失敗時該怎麼做。您可以指定是否等待元件報告他們可以更新。如果裝置套用失敗的部署，您也可以指定是否要將裝置復原至先前的組態。

- 停止組態

定義停止部署的時間和方式。如果符合您定義的條件，部署會停止並失敗。例如，您可以設定部署，在最小數量的裝置接收後，有一定百分比的裝置無法套用該部署時停止。

- 推展組態

定義部署推展到目標裝置的速率。您可以使用最小和最大速率邊界來設定指數速率增加。

- 逾時組態

定義每個裝置必須套用部署的時間上限。如果裝置超過您指定的持續時間，則裝置無法套用部署。

⚠ Important

自訂元件可以在 S3 儲存貯體中定義成品。AWS IoT Greengrass 核心軟體部署元件時，會從下載元件的成品 AWS 雲端。根據預設，核心裝置角色不允許存取 S3 儲存貯體。若要部署定義 S3 儲存貯體中成品的自訂元件，核心裝置角色必須授予從該儲存貯體下載成品的許可。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

主題

- [建立部署](#)
- [建立子部署](#)

- [修訂部署](#)
- [取消部署](#)
- [檢查部署狀態](#)

建立部署

您可以建立以物件或物件群組為目標的部署。

當您建立部署時，您可以設定要部署的軟體元件，以及部署任務如何推展到目標裝置。您可以在您提供的 JSON 檔案中定義部署 AWS CLI。

部署目標會決定您要在其中執行元件的裝置。若要部署到一個核心裝置，請指定物件。若要部署到多個核心裝置，請指定包含這些裝置的物件群組。如需如何設定物件群組的詳細資訊，請參閱《AWS IoT 開發人員指南》中的[靜態物件群組](#)和[動態物件群組](#)。

請依照本節中的步驟來建立目標的部署。如需如何更新具有部署之目標上的軟體元件的詳細資訊，請參閱[修訂部署](#)。

Warning

[CreateDeployment](#) 操作可以從核心裝置解除安裝元件。如果元件存在於先前的部署中，而非新的部署中，核心裝置會解除安裝該元件。為了避免解除安裝元件，請先使用 [ListDeployments](#) 操作來檢查部署的目標是否已有現有的部署。然後，當您建立新的部署時，請使用 [GetDeployment](#) 操作從該現有部署開始。

建立部署 (AWS CLI)

1. 建立名為 `deployment.json` 的檔案，然後將下列 JSON 物件複製到檔案。將 `targetArn` 取代為要用於部署之 AWS IoT 物件或物件群組的 ARN。物件和物件群組 ARNs 的格式如下：
 - 物件：`arn:aws:iot:region:account-id:thing/thingName`
 - 物件群組：`arn:aws:iot:region:account-id:thinggroup/thingGroupName`

```
{
  "targetArn": "targetArn"
}
```

2. 檢查部署目標是否有您要修改的現有部署。請執行下列操作：

- a. 執行下列命令來列出部署目標的部署。將 *targetArn* 取代為目標 AWS IoT 物件或物件群組的 ARN。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含具有目標最新部署的清單。如果回應是空的，則目標沒有現有的部署，您可以跳到 [Step 3](#)。否則，*deploymentId* 請從回應中複製 以用於下一個步驟。

Note

您也可以修訂目標最新修訂版以外的部署。指定 `--history-filter ALL` 引數以列出目標的所有部署。然後，複製您要修改的部署 ID。

- b. 執行下列命令以取得部署的詳細資訊。這些詳細資訊包括中繼資料、元件和任務組態。將 *deploymentId* 取代為上一個步驟的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的詳細資訊。

- c. 將下列任一鍵/值對從上一個命令的回應複製到 `deployment.json`。您可以變更新部署的這些值。

- `deploymentName` – 部署的名稱。
- `components` – 部署的元件。若要解除安裝元件，請從此物件中移除它。
- `deploymentPolicies` – 部署的政策。
- `iotJobConfiguration` – 部署的任務組態。
- `tags` – 部署的標籤。

3. (選用) 定義部署的名稱。將 *deploymentName* 取代為部署的名稱。

```
{  
  "targetArn": "targetArn",  
  "deploymentName": "deploymentName"  
}
```

4. 新增每個元件以部署目標裝置。若要這樣做，請將索引鍵/值對新增至components物件，其中索引鍵是元件名稱，而值是包含該元件詳細資訊的物件。為您新增的每個元件指定下列詳細資訊：

- version – 要部署的元件版本。
- configurationUpdate – 要部署的[組態更新](#)。更新是一種修補程式操作，可修改元件在每個目標裝置上的現有組態，如果元件不存在於目標裝置上，則會修改元件的預設組態。您可以指定下列組態更新：
 - 重設更新 (reset) – (選用) JSON 指標清單，定義要重設為目標裝置上的預設值的組態值。AWS IoT Greengrass 核心軟體會在套用合併更新之前套用重設更新。如需詳細資訊，請參閱[重設更新](#)。
 - 合併更新 (merge) – (選用) 定義要合併到目標裝置的組態值的 JSON 文件。您必須將 JSON 文件序列化為字串。如需詳細資訊，請參閱[合併更新](#)。
- runWith – (選用) AWS IoT Greengrass 核心軟體用來在核心裝置上執行此元件程序的系統程序選項。如果您省略runWith物件中的參數，AWS IoT Greengrass Core 軟體會使用您在[Greengrass 核元件](#)上設定的預設值。

您可以指定下列任一選項：

- posixUser – POSIX 系統使用者和選用的群組，用於在 Linux 核心裝置上執行此元件。如果指定，使用者和群組必須存在於每個 Linux 核心裝置上。以下列格式指定使用者和群組，並以冒號(:) 分隔：user:group。群組為選用項目。如果您未指定群組，則 AWS IoT Greengrass Core 軟體會使用使用者的主要群組。如需詳細資訊，請參閱[設定執行元件的使用者](#)。
- windowsUser – 用來在 Windows 核心裝置上執行此元件的 Windows 使用者。使用者必須存在於每個 Windows 核心裝置上，且其名稱和密碼必須存放在 LocalSystem 帳戶的 Credentials Manager 執行個體中。如需詳細資訊，請參閱[設定執行元件的使用者](#)。

此功能適用於 [Greengrass 核元件](#) 的 v2.5.0 和更新版本。

- systemResourceLimits – 要套用至此元件程序的系統資源限制。您可以將系統資源限制套用至一般和非容器 Lambda 元件。如需詳細資訊，請參閱[設定元件的系統資源限制](#)。

您可以指定下列任一選項：

- cpus – 此元件程序可在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定為 2，將此元件的程序限制為每個 CPU 核心的 50% 用量。在具有 1 個 CPU 核心的裝置上，您可以將此值設定為 0.25，以將此元件的程序限制為 CPU 的 25% 用量。如果您將此值設定為

大於 CPU 核心數量的數字，則 AWS IoT Greengrass Core 軟體不會限制元件的 CPU 使用量。

- memory – 此元件的程序可在核心裝置上使用的 RAM 數量上限（以 KB 為單位）。

此功能適用於 2.4.0 版及更新版本的 [Greengrass 核心元件](#)。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

Example範例基本組態更新

下列範例components物件指定 來部署元件 com.example.PythonRuntime，該元件預期組態參數名為 pythonVersion。

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Example具有重設和合併更新的範例組態更新

請考慮com.example.IndustrialDashboard具有下列預設組態的範例工業儀表板元件。

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
}
```

```
"tags": []
}
```

下列組態更新會指定下列指示：

1. 將 HTTPS 設定重設為其預設值 (true)。
2. 將工業標籤清單重設為空白清單。
3. 合併工業標籤清單，以識別兩個鍋爐的溫度和壓力資料串流。

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
}
```

下列範例components物件會指定 部署此工業儀表板元件和組態更新。

```
{
  "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. (選用) 定義部署的部署政策。您可以設定核心裝置何時可以安全地套用部署，或如果核心裝置無法套用部署時該怎麼辦。若要這麼做，請將 `deploymentPolicies` 物件新增至 `deployment.json`，然後執行下列任何動作：

1. (選用) 指定元件更新政策 (`componentUpdatePolicy`)。此政策會定義部署是否允許元件延遲更新，直到準備好更新為止。例如，元件可能需要清除資源或完成關鍵動作，才能重新啟動以套用更新。此政策也會定義元件必須回應更新通知的時間量。

此政策是具有下列參數的物件：

- `action` – (選用) 是否要通知元件，並等待元件準備好更新時回報。您可以從以下選項中選擇：
 - `NOTIFY_COMPONENTS` – 部署會在停止前向每個元件發出通知並更新該元件。元件可以使用 [SubscribeToComponentUpdates](#) IPC 操作來接收這些通知。
 - `SKIP_NOTIFY_COMPONENTS` – 部署不會向元件發出通知，也不會等到可安全更新元件時才更新。

預設為 `NOTIFY_COMPONENTS`。

- `timeoutInSeconds` 每個元件必須使用 [DeferComponentUpdate](#) IPC 操作回應更新通知的時間，以秒為單位。如果元件未在此時間內回應，則部署會在核心裝置上繼續。

預設為 60 秒。

2. (選用) 指定組態驗證政策 (`configurationValidationPolicy`)。

此政策定義每個元件必須驗證部署中組態更新的時間長度。元件可以使用

[SubscribeToValidateConfigurationUpdates](#) IPC 操作來訂閱通知，以取得自己的組態更新。

然後，元件可以使用 [SendConfigurationValidityReport](#) IPC 操作來告知 AWS IoT Greengrass Core 軟體組態更新是否有效。如果組態更新無效，部署會失敗。

此政策是具有下列參數的物件：

- `timeoutInSeconds` (選用) 每個元件驗證組態更新所需的秒數。如果元件未在此時間內回應，則部署會在核心裝置上繼續。

預設為 30 秒。

3. (選用) 指定失敗處理政策 (`failureHandlingPolicy`)。此政策是字串，可定義部署失敗時是否要復原裝置。您可以從以下選項中選擇：

- ROLLBACK – 如果核心裝置上的部署失敗，則 AWS IoT Greengrass 核心軟體會將該核心裝置復原為先前的組態。
- DO_NOTHING – 如果在核心裝置上部署失敗，則 AWS IoT Greengrass 核心軟體會保留新的組態。如果新組態無效，這可能會導致元件損壞。

預設為 ROLLBACK。

您在 中的部署 `deployment.json` 可能看起來類似下列範例：

```
{
  "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. (選用) 定義部署如何停止、推展或逾時。AWS IoT Greengrass AWS IoT Core 使用任務將部署傳送至核心裝置，因此這些選項與 AWS IoT Core 任務的組態選項相同。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的[任務推出和中止組態](#)。

若要定義任務選項，請將*iotJobConfiguration*物件新增至 *deployment.json*。然後，定義要設定的選項。

您在 中的部署*deployment.json*可能看起來類似下列範例：

```
{
  "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. (選用) 為部署新增標籤 (tags)。如需詳細資訊，請參閱[標記您的 AWS IoT Greengrass Version 2 資源](#)。
8. 執行下列命令，從 建立部署 deployment.json。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

回應包含可識別此部署 deploymentId 的。您可以使用部署 ID 來檢查部署的狀態。如需詳細資訊，請參閱[檢查部署狀態](#)。

更新元件組態

元件組態是定義每個元件參數的 JSON 物件。每個元件的配方都會定義其預設組態，當您將元件部署到核心裝置時，會對其進行修改。

建立部署時，您可以指定要套用至每個元件的組態更新。組態更新是修補程式操作，這表示更新會修改核心裝置上存在的元件組態。如果核心裝置沒有元件，則組態更新會修改並套用該部署的預設組態。

組態更新會定義重設更新和合併更新。重設更新會定義要重設為其預設值或移除的組態值。合併更新會定義要為元件設定的新組態值。部署組態更新時，AWS IoT Greengrass 核心軟體會在合併更新之前執行重設更新。

元件可以驗證您部署的組態更新。當部署變更其組態時，元件會訂閱以接收通知，而且可以拒絕不支援的組態。如需詳細資訊，請參閱[與元件組態互動](#)。

主題

- [重設更新](#)
- [合併更新](#)
- [範例](#)

重設更新

重設更新會定義要重設為核心裝置上的預設值的組態值。如果組態值沒有預設值，則重設更新會從元件的組態中移除該值。這可協助您修正因無效的組態而中斷的元件。

使用 JSON 指標清單來定義要重設的組態值。JSON 指標以正斜線 開頭/。若要識別巢狀元件組態中的值，請使用正斜線 (/) 來分隔組態中每個層級的索引鍵。如需詳細資訊，請參閱 [JSON 指標規格](#)。

Note

您只能將整個清單重設為其預設值。您無法使用重設更新來重設清單中的個別元素。

若要將元件的整個組態重設為其預設值，請指定單一空字串做為重設更新。

```
"reset": [""]
```

合併更新

合併更新會定義要插入核心元件組態的組態值。合併更新是 AWS IoT Greengrass Core 軟體在重設重設更新中指定路徑中的值後，會合併的 JSON 物件。使用 AWS CLI 或 AWS SDKs 時，您必須將此 JSON 物件序列化為字串。

您可以合併不存在於元件預設組態中的鍵/值對。您也可以合併與具有相同索引鍵的值類型不同的索引鍵/值對。新值會取代舊值。這表示您可以變更組態物件的結構。

您可以合併 null 值和空字串、清單和物件。

Note

您無法將合併更新用於將元素插入或附加至清單。您可以取代整個清單，也可以定義每個元素都有唯一索引鍵的物件。

AWS IoT Greengrass 使用 JSON 做為組態值。JSON 會指定數字類型，但不會區分整數和浮點數。因此，組態值可能會轉換為浮點數 AWS IoT Greengrass。為了確保您的元件使用正確

的資料類型，建議您將數值組態值定義為字串。然後，讓您的元件將它們剖析為整數或浮點數。這可確保您的組態值在組態和核心裝置上具有相同的類型。

在合併更新中使用配方變數

此功能適用於 [Greengrass 核元件](#) 的 v2.6.0 和更新版本。

如果您將 Greengrass 核的 [interpolateComponentConfiguration](#) 組態選項設定為 true，您可以在合併更新中使用配方變數以外的 `component_dependency_name:configuration:json_pointer` 配方變數。例如，您可以在合併更新中使用 {iot:thingName} 配方變數，在元件組態值中包含核心裝置的 AWS IoT 物件名稱，例如 [程序間通訊 \(IPC\) 授權政策](#)。

範例

下列範例示範具有下列預設組態之儀表板元件的組態更新。此範例元件會顯示工業設備的相關資訊。

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

工業儀表板元件配方

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範例 1：合併更新

您可以建立套用下列組態更新的部署，這會指定合併更新，但不會指定重設更新。此組態更新會通知元件在 HTTP 連接埠 8080 上顯示儀表板，其中包含來自兩個沸騰器的資料。

Console

要合併的組態

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

```
}
```

AWS CLI

下列命令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

dashboard-deployment.json 檔案包含下列 JSON 文件。

```
{
  "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

下列 [Greengrass CLI](#) 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

dashboard-configuration.json 檔案包含下列 JSON 文件。

```
{
  "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"
    ]
  }
}
```

在此更新之後，儀表板元件具有下列組態。

```
{
  "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 範例 2：重設和合併更新

然後，您可以建立套用下列組態更新的部署，指定重設更新和合併更新。這些更新指定在預設 HTTPS 連接埠上顯示儀表板，其中包含來自不同沸騰器的資料。這些更新會修改先前範例中組態更新所產生的組態。

Console

重設路徑

```
[
  "/network/useHttps",
  "/tags"
]
```

要合併的組態

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

下列命令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file:///dashboard-
deployment2.json
```

dashboard-deployment2.json 檔案包含下列 JSON 文件。

```
{
  "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

下列 [Greengrass CLI](#) 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

dashboard-configuration2.json 檔案包含下列 JSON 文件。

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
  }
}
```

在此更新之後，儀表板元件具有下列組態。

```
{
  "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",
]
}
```

建立子部署

Note

子部署功能可在 Greengrass nucleus 2.9.0 版及更新版本上使用。無法將組態部署到具有舊版 Greengrass 核元件的子部署。

子部署是針對父部署中較小裝置子集的部署。您可以使用子部署將組態部署到較小的裝置子集。您也可以建立子部署，在該父部署中的一或多個裝置失敗時重試失敗的父部署。使用此功能，您可以選取在該父部署中失敗的裝置，並建立子部署以測試組態，直到子部署成功為止。子部署成功後，您可以將該組態重新部署到父部署。

遵循本節中的步驟來建立子部署並檢查其狀態。如需如何建立部署的詳細資訊，請參閱[建立部署](#)。

建立子部署 (AWS CLI)

1. 執行下列命令來擷取物件群組的最新部署。將命令中的 ARN 取代為要查詢之物件群組的 ARN。--history-filter 設定為 **LATEST_ONLY** 以查看該物件群組的最新部署。

```
aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
```

2. 將 deploymentId 從回應複製到 list-deployments 命令，以在下一個步驟中使用。
3. 執行下列命令來擷取部署的狀態。deploymentId 將取代為要查詢之部署的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

4. 將 `iotJobId` 從回應複製到 `get-deployment` 命令，以在下列步驟中使用。
5. 執行下列命令，以擷取指定任務的任務執行清單。將 *jobID* 取代為上一個步驟 `iotJobId` 的。
將 *##* 取代為您要篩選的狀態。您可以篩選具有下列狀態的結果：

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED
- TIMED_OUT
- REJECTED
- REMOVED
- CANCELED

```
aws iot list-job-executions-for-job --job-id jobID --status status
```

6. 為您的子部署建立新的 AWS IoT 物件群組，或使用現有的物件群組。然後，將 AWS IoT 物件新增至此物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的機群。當您將軟體元件部署到裝置時，您可以鎖定個別裝置或裝置群組。您可以將裝置新增至具有作用中 Greengrass 部署的物件群組。新增後，您就可以將該物件群組的軟體元件部署到該裝置。

若要建立新的物件群組並將您的裝置新增至其中，請執行下列動作：

- a. 建立 AWS IoT 物件群組。將 *MyGreengrassCoreGroup* 取代為新物件群組的名稱。您無法在物件群組名稱中使用冒號 (:)。

 Note

如果子部署的物件群組與一個搭配使用 `parentTargetArn`，則無法與不同的父機群重複使用。如果物件群組已用於為另一個機群建立子部署，則 API 會傳回錯誤。

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，回應看起來與下列範例類似：

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. 將佈建的 Greengrass 核心新增至您的物件群組。使用這些參數執行下列命令：

- 將 *MyGreengrassCore* 取代為您佈建的 Greengrass 核心的名稱。
- 將 *MyGreengrassCoreGroup* 取代為您的物件群組名稱。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，命令不會有任何輸出。

7. 建立名為 `deployment.json` 的檔案，然後將下列 JSON 物件複製到檔案。將 *targetArn* 取代為 AWS IoT 物件群組的 ARN，以鎖定子部署的目標。子部署目標只能是物件群組。物件群組 ARNs 的格式如下：

- 物件群組 – `arn:aws:iot:region:account-id:thinggroup/thingGroupName`

```
{
  "targetArn": "targetArn"
}
```

8. 再次執行下列命令，以取得原始部署的詳細資訊。這些詳細資訊包括中繼資料、元件和任務組態。將 *deploymentId* 取代為來自的 ID [Step 1](#)。您可以使用此部署組態來設定子部署，並視需要進行變更。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的詳細資訊。將下列任一鍵值對從 `get-deployment` 命令的回應複製到 `deployment.json`。您可以變更子部署的這些值。如需此命令詳細資訊的詳細資訊，請參閱 [GetDeployment](#)。

- components – 部署的元件。若要解除安裝元件，請從此物件中移除它。
 - deploymentName – 部署的名稱。
 - deploymentPolicies – 部署的政策。
 - iotJobConfiguration – 部署的任務組態。
 - parentTargetArn – 父系部署的目標。
 - tags – 部署的標籤。
9. 執行下列命令，從 建立子部署deployment.json。將 *subdeploymentName* 取代為子部署的名稱。

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-json file://deployment.json
```

回應包含可識別此子部署deploymentId的。您可以使用部署 ID 來檢查部署的狀態。如需詳細資訊，請參閱[檢查部署狀態](#)。

10. 如果子部署成功，您可以使用其組態來修改父部署。複製deployment.json您在上一個步驟中使用的。targetArn 將 JSON 檔案中的 取代為父系部署的 ARN，並執行下列命令來使用此新組態建立父系部署。

Note

如果您建立新的父機群部署修訂版，它會取代該父部署的所有部署修訂版和子部署。如需詳細資訊，請參閱[修訂部署](#)。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

回應包含可識別此部署deploymentId的。您可以使用部署 ID 來檢查部署的狀態。如需詳細資訊，請參閱[檢查部署狀態](#)。

修訂部署

每個目標物件或物件群組一次可以有一個作用中的部署。當您為已經有部署的目標建立部署時，新部署中的軟體元件會取代先前部署的軟體元件。如果新部署未定義先前部署定義的元件，則 AWS IoT

Greengrass Core 軟體會從目標核心裝置移除該元件。您可以修改現有的部署，以免將核心裝置上執行的元件從先前的部署移除到目標。

若要修改部署，您可以建立從先前部署中存在的相同元件和組態開始的部署。您可以使用 [CreateDeployment](#) 操作，這是您用來[建立部署](#)的相同操作。

修改部署 (AWS CLI)

1. 執行下列命令來列出部署目標的部署。將 *targetArn* 取代為目標 AWS IoT 物件或物件群組的 ARN。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含具有目標最新部署的清單。deploymentId 從回應中複製 以用於下一個步驟。

Note

您也可以修改目標最新修訂版以外的部署。指定 `--history-filter ALL` 引數以列出目標的所有部署。然後，複製您要修改的部署 ID。

2. 執行下列命令以取得部署的詳細資訊。這些詳細資訊包括中繼資料、元件和任務組態。將 *deploymentId* 取代為上一個步驟的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的詳細資訊。

3. 建立名為 `deployment.json` 的文件，並將前一個命令的回應複製到檔案中。
4. 從 JSON 物件的 `deployment.json` 中移除以下鍵值組：

- deploymentId
- revisionId
- iotJobId
- iotJobArn
- creationTimestamp
- isLatestForTarget
- deploymentStatus

[CreateDeployment](#) 操作預期承載具有下列結構。

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. 在 deployment.json 中，執行下列任何一項：

- 變更部署的名稱 (deploymentName)。
- 變更部署的元件 (components)。
- 變更部署的政策 (deploymentPolicies)。
- 變更部署的任務組態 (iotJobConfiguration)。
- 變更部署的標籤 (tags)。

如需如何定義這些部署詳細資訊的詳細資訊，請參閱[建立部署](#)。

6. 執行下列命令，從 建立部署 deployment.json。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

回應包含可識別此部署 deploymentId 的。您可以使用部署 ID 來檢查部署的狀態。如需詳細資訊，請參閱[檢查部署狀態](#)。

取消部署

您可以取消作用中部署，以防止其軟體元件安裝在 AWS IoT Greengrass 核心裝置上。如果您取消以物件群組為目標的部署，您新增至群組的核心裝置將不會收到該持續部署。如果核心裝置已執行部署，則當您取消部署時，不會變更該裝置上的元件。您必須[建立新的部署或修改部署](#)，以修改在接收已取消部署的核心裝置上執行的元件。

取消部署 (AWS CLI)

1. 執行下列命令來尋找目標的最新部署修訂的 ID。最新修訂是唯一可以對目標啟用的部署，因為先前部署會在您建立新的修訂時取消。將 *targetArn* 取代為目標 AWS IoT 物件或物件群組的 ARN。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含具有目標最新部署的清單。deploymentId 從回應中複製 以用於下一個步驟。

2. 執行下列命令來取消部署。將 *deploymentId* 取代為上一個步驟的 ID。

```
aws greengrassv2 cancel-deployment --deployment-id deploymentId
```

如果操作成功，部署狀態會變更為 CANCELED。

檢查部署狀態

您可以檢查您在其中建立的部署狀態 AWS IoT Greengrass。您也可以檢查將部署推展到每個核心裝置 AWS IoT 的任務狀態。當部署處於作用中狀態時，AWS IoT 任務的狀態為 IN_PROGRESS。建立新的部署修訂版之後，上一個修訂版 AWS IoT 的任務狀態會變更為 CANCELLED。

主題

- [檢查部署狀態](#)
- [檢查裝置部署狀態](#)

檢查部署狀態

您可以檢查依目標或其 ID 識別的部署狀態。

依目標檢查部署狀態 (AWS CLI)

- 執行下列命令，以擷取目標的最新部署狀態。將 *targetArn* 取代為部署目標 AWS IoT 之實物或實物群組的 Amazon Resource Name (ARN)。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含具有目標最新部署的清單。此部署物件包含部署的狀態。

依 ID 檢查部署狀態 (AWS CLI)

- 執行下列命令來擷取部署的狀態。將 *deploymentId* 取代為要查詢的部署 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的狀態。

檢查裝置部署狀態

您可以檢查套用至個別核心裝置的部署任務狀態。您也可以檢查物件群組部署的部署任務狀態。

檢查核心裝置的部署任務狀態 (AWS CLI)

- 執行下列命令，以擷取核心裝置的所有部署任務狀態。將 *coreDeviceName* 取代為要查詢的核心裝置名稱。

```
aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
```

回應包含核心裝置的部署任務清單。您可以透過任務的 *deploymentId* 或 *targetArn* 來識別部署的任務。每個部署任務都包含核心裝置上任務的狀態。

檢查物件群組的部署狀態 (AWS CLI)

- 執行下列命令來擷取現有部署的 ID。將 *targetArn* 取代為目標物件群組的 ARN。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

回應包含具有目標最新部署的清單。*deploymentId* 從回應中複製 以用於下一個步驟。

Note

您也可以列出目標最新部署以外的部署。指定引 `--history-filter ALL` 數以列出目標的所有部署。然後，複製您要檢查狀態的部署 ID。

- 執行下列命令以取得部署的詳細資訊。將 *deploymentID* 取代為上一個步驟的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

回應包含部署的相關資訊。iotJobId 從回應中複製 以用於下列步驟。

3. 執行下列命令來描述核心裝置在部署時的任務執行。將 *iotJobId* 和 *coreDeviceThingName* 取代為上一個步驟的任務 ID，以及您要檢查狀態的核心裝置。

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

回應包含核心裝置部署任務執行的狀態，以及狀態的詳細資訊。detailsMap 包含下列資訊：

- detailed-deployment-status – 部署結果狀態，可以是下列其中一個值：
 - SUCCESSFUL – 部署成功。
 - FAILED_NO_STATE_CHANGE – 核心裝置準備套用部署時，部署失敗。
 - FAILED_ROLLBACK_NOT_REQUESTED – 部署失敗，而且部署未指定 轉返先前的工作組態，因此核心裝置可能無法正常運作。
 - FAILED_ROLLBACK_COMPLETE – 部署失敗，核心裝置成功復原至先前的工作組態。
 - FAILED_UNABLE_TO_ROLLBACK – 部署失敗，且核心裝置無法復原至先前的工作組態，因此核心裝置可能無法正常運作。

如果部署失敗，請檢查 deployment-failure-cause 值和核心裝置的日誌檔案，以識別問題。如需如何存取核心裝置日誌檔案的詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

- deployment-failure-cause – 錯誤訊息，提供任務執行失敗原因的其他詳細資訊。

回應看起來與下列範例類似。

```
{
  "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 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  
}  
}
```

在 中記錄和監控 AWS IoT Greengrass

監控是維護和 AWS 解決方案的可靠性、可用性 AWS IoT Greengrass 和效能的重要部分。您應該從 AWS 解決方案的所有部分收集監控資料，以便在發生多點失敗時更輕鬆地偵錯。開始監控之前 AWS IoT Greengrass，您應該建立監控計畫，其中包含下列問題的答案：

- 監控目標是什麼？
- 監控哪些資源？
- 監控這些資源的頻率為何？
- 將使用哪些監控工具？
- 誰將執行監控任務？
- 發生問題時應該通知誰？

主題

- [監控工具](#)
- [監控 AWS IoT Greengrass 日誌](#)
- [使用 記錄 AWS IoT Greengrass V2 API 呼叫 AWS CloudTrail](#)
- [從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)
- [取得部署和元件運作狀態通知](#)
- [檢查 Greengrass 核心裝置狀態](#)

監控工具

AWS 提供可用來監控的工具 AWS IoT Greengrass。您可設定部分這些工具以為您執行監控工作。部分工具將需要手動操作。建議您盡可能自動化監控任務。

您可以使用下列自動化監控工具來監控 AWS IoT Greengrass 和報告問題：

- Amazon CloudWatch Logs – 從 AWS CloudTrail 或其他來源監控、存放和存取您的日誌檔案。如需詳細資訊，請參閱《Amazon CloudWatch 使用者指南》中的[監控日誌檔案](#)。
- AWS CloudTrail 日誌監控 – 在帳戶之間共用日誌檔案、透過將日誌檔案傳送至 CloudWatch Logs 即時監控 CloudTrail 日誌檔案、在 Java 中寫入日誌處理應用程式，以及驗證您的日誌檔案在 CloudTrail 交付後並未變更。如需詳細資訊，請參閱 AWS CloudTrail 使用者指南中的[使用 CloudTrail 日誌檔案](#)。

- Greengrass 系統運作狀態遙測 – 訂閱以接收從 Greengrass 核心傳送的遙測資料。如需詳細資訊，請參閱[the section called “收集系統運作狀態遙測資料”](#)。
- 裝置運作狀態通知 使用 Amazon EventBridge 建立事件，以接收有關部署和元件的狀態更新。如需詳細資訊，請參閱[取得部署和元件運作狀態通知](#)。
- 機群狀態服務 – 使用機群狀態 API 操作來檢查核心裝置及其 Greengrass 元件的狀態。您也可以可以在 AWS IoT Greengrass 主控台中檢視機群狀態資訊。如需詳細資訊，請參閱[檢查 Greengrass 核心裝置狀態](#)。

監控 AWS IoT Greengrass 日誌

AWS IoT Greengrass 包含 雲端服務和 AWS IoT Greengrass 核心軟體。AWS IoT Greengrass 核心軟體可以將日誌寫入 Amazon CloudWatch Logs 和核心裝置的本機檔案系統。在核心裝置上執行的 Greengrass 元件也可以將日誌寫入 CloudWatch Logs 和本機檔案系統。您可以使用日誌來監控事件和排解疑難問題。所有 AWS IoT Greengrass 日誌項目都包含時間戳記、日誌層級和事件的相關資訊。

根據預設，AWS IoT Greengrass 核心軟體只會將日誌寫入本機檔案系統。您可以即時檢視檔案系統日誌，以便偵錯您開發和部署的 Greengrass 元件。您也可以設定核心裝置將日誌寫入 CloudWatch Logs，以便對核心裝置進行故障診斷，而無需存取本機檔案系統。如需詳細資訊，請參閱[啟用記錄至 CloudWatch Logs](#)。

主題

- [存取檔案系統日誌](#)
- [存取 CloudWatch Logs](#)
- [存取系統服務日誌](#)
- [啟用記錄至 CloudWatch Logs](#)
- [設定的記錄 AWS IoT Greengrass](#)
- [AWS CloudTrail 日誌](#)

存取檔案系統日誌

Core AWS IoT Greengrass 軟體會將日誌存放在核心裝置上的 `/greengrass/v2/logs` 資料夾中，其中 `/greengrass/v2` 是 AWS IoT Greengrass 根資料夾的路徑。日誌資料夾具有下列結構。

```
/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` – AWS IoT Greengrass 核心軟體日誌檔案。使用此日誌檔案來檢視元件和部署的即時資訊。此日誌檔案包含 Greengrass 核的日誌，這是 AWS IoT Greengrass Core 軟體的核心，以及外掛程式元件，例如 [日誌管理員](#) 和 [秘密管理員](#)。
- `ComponentName.log` – Greengrass 元件日誌檔案。使用元件日誌檔案來檢視在核心裝置上執行之 Greengrass 元件的即時資訊。一般元件和 Lambda 元件會將標準輸出 (stdout) 和標準錯誤 (stderr) 寫入這些日誌檔案。
- `main.log` – 處理元件生命週期之 main 服務的日誌檔案。此日誌檔案一律為空白。

如需外掛程式、一般元件和 Lambda 元件之間差異的詳細資訊，請參閱 [元件類型](#)。

使用檔案系統日誌時有下列考量：

- 根使用者許可

您必須具有讀取檔案系統 AWS IoT Greengrass 日誌的根許可。

- 日誌檔案輪換

AWS IoT Greengrass 核心軟體每小時輪換日誌檔案，或在超過檔案大小限制時輪換日誌檔案。輪換的日誌檔案在檔案名稱中包含時間戳記。例如，輪換 AWS IoT Greengrass 的核心軟體日誌檔可能命名為 `greengrass_2021_09_14_15_0.log`。預設檔案大小限制為 1,024 KB (1 MB)。您可以在 [Greengrass 核元件](#) 上設定檔案大小限制。

- 日誌檔案刪除

AWS IoT Greengrass 核心軟體會在 AWS IoT Greengrass 核心軟體日誌檔案或 Greengrass 元件日誌檔案的大小超過磁碟空間限制時，清除先前的日誌檔案，包括輪換的日誌檔案。Core AWS IoT Greengrass 軟體日誌和每個元件日誌的預設磁碟空間限制為 10,240 KB (10 MB)。您可以在 [Greengrass 核元件](#) 或日誌管理員元件上設定 AWS IoT Greengrass 核心軟體日誌磁碟空間限制。
[???您可以在日誌管理員元件上設定每個元件的日誌磁碟空間限制。](#)

檢視 AWS IoT Greengrass 核心軟體日誌檔案

- 執行下列命令來即時檢視日誌檔案。`/greengrass/v2` 將取代為 AWS IoT Greengrass 根資料夾的路徑。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

type 命令會將檔案的內容寫入終端機。多次執行此命令，以觀察 檔案中的變更。

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

檢視元件的日誌檔案

- 執行下列命令來即時檢視日誌檔案。將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 AWS IoT Greengrass 根資料夾的路徑，並將 `com.example.HelloWorld` 取代為元件名稱。

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
```

您也可以使用 [Greengrass CLI](#) 的 logs 命令來分析核心裝置上的 Greengrass 日誌。若要使用 logs 命令，您必須設定 [Greengrass 核](#) 以輸出 JSON 格式日誌檔案。如需詳細資訊，請參閱 [Greengrass 命令列界面](#) 及 [日誌](#)。

存取 CloudWatch Logs

您可以部署 [日誌管理員元件](#)，將核心裝置設定為寫入 CloudWatch Logs。如需詳細資訊，請參閱 [啟用記錄至 CloudWatch Logs](#)。然後，您可以在 Amazon CloudWatch 主控台的日誌頁面上或使用 CloudWatch Logs API 檢視日誌。

日誌群組名稱

```
/aws/greengrass/componentType/region/componentName
```

日誌群組名稱使用下列變數：

- *componentType* – 元件的類型，可以是下列其中一項：
 - *GreengrassSystemComponent* – 此日誌群組包含核心和外掛程式元件的日誌，這些元件在與 Greengrass 核心相同的 JVM 中執行。元件是 [Greengrass 核](#)的一部分。
 - *UserComponent* – 此日誌群組包含一般元件、Lambda 元件和裝置上的其他應用程式的日誌。元件不屬於 Greengrass 核。

如需詳細資訊，請參閱[元件類型](#)。

- *region* – 核心裝置使用的 AWS 區域。
- *componentName* – 元件的名稱。對於系統日誌，此值為 System。

日誌串流名稱

```
/date/thing/thingName
```

日誌串流名稱使用下列變數：

- *date* – 日誌的日期，例如 2020/12/15。日誌管理員元件使用 yyyy/MM/dd 格式。
- *thingName* – 核心裝置的名稱。

Note

如果物件名稱包含冒號 (:)，則日誌管理員會以加號 () 取代冒號+。

當您使用日誌管理員元件寫入 CloudWatch Logs 時，適用下列考量：

- 日誌延遲

Note

我們建議您升級至日誌管理員 2.3.0 版，以減少輪換和作用中日誌檔案的日誌延遲。當您升級至日誌管理員 2.3.0 時，我們建議您也升級至 Greengrass 核 2.9.1。

日誌管理員元件 2.2.8 版（及更舊版本）只會從輪換的日誌檔案處理和上傳日誌。根據預設，AWS IoT Greengrass 核心軟體會每小時輪換日誌檔案，或在 1,024 KB 之後輪換日誌檔案。因此，日誌管理員元件只會在 AWS IoT Greengrass Core 軟體或 Greengrass 元件寫入超過 1,024 KB 的日誌之後上傳日誌。您可以設定較低的日誌檔案大小限制，讓日誌檔案更頻繁地輪換。這會導致日誌管理員元件更頻繁地將日誌上傳至 CloudWatch Logs。

日誌管理員元件 2.3.0 版（及更新版本）會處理和上傳所有日誌。當您撰寫新的日誌時，日誌管理員 2.3.0 版（及更新版本）會處理並直接上傳該作用中日誌檔案，而不是等待輪換。這表示您可以在 5 分鐘內檢視新日誌。

日誌管理員元件會定期上傳新日誌。根據預設，日誌管理員元件會每 5 分鐘上傳新日誌。您可以設定較低的上傳間隔，讓日誌管理員元件透過設定 `LogPeriodicUploadIntervalSec`，更頻繁地將日誌上傳至 CloudWatch Logs。如需如何設定此定期間隔的詳細資訊，請參閱[組態](#)。

日誌可以近乎即時地從相同的 Greengrass 檔案系統上傳。如果您需要即時觀察日誌，請考慮使用[檔案系統日誌](#)。

Note

如果您使用不同的檔案系統來寫入日誌，日誌管理員會還原至日誌管理員元件 2.2.8 版及更早版本中的行為。如需有關存取檔案系統日誌的資訊，請參閱[存取檔案系統日誌](#)。

• 時鐘扭曲

日誌管理員元件使用標準 Signature 第 4 版簽署程序來建立對 CloudWatch Logs 的 API 請求。如果核心裝置上的系統時間不同步超過 15 分鐘，則 CloudWatch Logs 會拒絕請求。如需詳細資訊，請參閱 AWS 一般參考 中的 [Signature 第 4 版簽署程序](#)。

存取系統服務日誌

如果您將 [AWS IoT Greengrass Core 軟體設定為系統服務](#)，您可以檢視系統服務日誌來疑難排解問題，例如軟體無法啟動。

檢視系統服務日誌 (CLI)

1. 執行下列命令以檢視 AWS IoT Greengrass Core 軟體系統服務日誌。

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. 在 Windows 裝置上，AWS IoT Greengrass 核心軟體會為系統服務錯誤建立個別的日誌檔案。執行下列命令以檢視系統服務錯誤日誌。

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\greengrass.err.log
```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.err.log
```

在 Windows 裝置上，您也可以使用事件檢視器應用程式來檢視系統服務日誌。

檢視 Windows 服務日誌（事件檢視器）

1. 開啟事件檢視器應用程式。
2. 選取 Windows Logs 以展開日誌。
3. 選擇應用程式以檢視應用程式服務日誌。
4. 尋找並開啟來源為 的事件日誌greengrass。

啟用記錄至 CloudWatch Logs

您可以部署[日誌管理員元件](#)，以設定核心裝置將日誌寫入 CloudWatch Logs。您可以啟用 CloudWatch Logs for AWS IoT Greengrass Core 軟體日誌，也可以針對特定 Greengrass 元件啟用 CloudWatch Logs。

Note

Greengrass 核心裝置的字符交換角色必須允許核心裝置寫入 CloudWatch Logs，如下列範例 IAM 政策所示。如果您[使用自動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)，則您的核心裝置具有這些許可。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

若要設定核心裝置將 AWS IoT Greengrass 核心軟體日誌寫入 CloudWatch Logs，[請建立部署](#)，指定針對 `aws.greengrass.LogManager` 元件 `uploadToCloudWatch` 設定為 `true` 的組態更新。AWS IoT Greengrass 核心軟體日誌包含 [Greengrass 核](#)和[外掛程式元件](#)的日誌。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

若要設定核心裝置將 Greengrass 元件的日誌寫入 CloudWatch Logs，[請建立部署](#)，指定將元件新增至元件記錄組態清單的組態更新。當您將元件新增至此清單時，日誌管理員元件會將其日誌寫入 CloudWatch Logs。元件日誌包含[一般元件](#)和 [Lambda 元件](#)的日誌。

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
  }
}
```

部署日誌管理員元件時，您也可以設定磁碟空間限制，以及核心裝置是否在將日誌檔案寫入 CloudWatch Logs 後刪除日誌檔案。如需詳細資訊，請參閱[設定的記錄 AWS IoT Greengrass](#)。

設定的記錄 AWS IoT Greengrass

您可以設定下列選項來自訂 Greengrass 核心裝置的記錄。若要設定這些選項，[請建立指定 Greengrass 核或日誌管理員元件組態更新的部署](#)。

- 將日誌寫入 CloudWatch Logs

若要遠端疑難排解核心裝置，您可以設定核心裝置將 AWS IoT Greengrass 核心軟體和元件日誌寫入 CloudWatch Logs。若要這樣做，請部署和設定[日誌管理員元件](#)。如需詳細資訊，請參閱[啟用記錄至 CloudWatch Logs](#)。

- 刪除上傳的日誌檔案

若要減少磁碟空間使用量，您可以將核心裝置設定為在將日誌檔案寫入 CloudWatch Logs 後刪除日誌檔案。如需詳細資訊，請參閱日誌管理員元件的 `deleteLogFileAfterCloudUpload` 參數，您可以為[AWS IoT Greengrass 核心軟體日誌](#)和[元件日誌](#)指定參數。

- 日誌磁碟空間限制

若要限制磁碟空間用量，您可以在核心裝置上設定每個日誌的最大磁碟空間，包括其輪換的日誌檔案。例如，您可以設定 `greengrass.log` 和 輪換 `greengrass.log` 檔案的最大合併磁碟空間。如需詳細資訊，請參閱 Greengrass 核元件的 `logging.totalLogsSizeKB` 參數和日誌管理員元件的 `diskSpaceLimit` 參數，您可以為[AWS IoT Greengrass 核心軟體日誌](#)和[元件日誌](#)指定參數。

- 日誌檔案大小限制

您可以設定每個日誌檔案的檔案大小上限。日誌檔案超過此檔案大小限制後，AWS IoT Greengrass 核心軟體會建立新的日誌檔案。[日誌管理員元件](#) 2.28 版（及更舊版本）只會將輪換的日誌檔案寫入 CloudWatch Logs，因此您可以指定較低的檔案大小限制，以更頻繁地將日誌寫入 CloudWatch Logs。日誌管理員元件 2.3.0 版（及更新版本）會處理和上傳所有日誌，而不是等待輪換。如需詳細資訊，請參閱 Greengrass 核元件的[日誌檔案大小限制參數](#) (logging.fileSizeKB)。

- 日誌層級下限

您可以設定 Greengrass 核元件寫入檔案系統日誌的最低日誌層級。例如，您可以指定DEBUG關卡日誌以協助故障診斷，也可以指定ERROR關卡日誌，以減少核心裝置建立的日誌數量。如需詳細資訊，請參閱 Greengrass 核元件的[日誌層級參數](#) (logging.level)。

您也可以設定日誌管理員元件寫入 CloudWatch Logs 的最低日誌層級。例如，您可以指定較高的日誌層級來降低[記錄成本](#)。如需詳細資訊，請參閱日誌管理員元件的 minimumLogLevel 參數，您可以為[AWS IoT Greengrass 核心軟體日誌](#)和[元件日誌](#)指定參數。

- 檢查日誌以寫入 CloudWatch Logs 的間隔

若要增加或減少日誌管理員元件將日誌寫入 CloudWatch Logs 的頻率，您可以設定其檢查要寫入新日誌檔案的間隔。例如，您可以指定比預設 5 分鐘間隔更快的間隔來檢視 CloudWatch Logs 中的日誌。您可以指定較高的間隔來降低成本，因為日誌管理員元件會將日誌檔案批次處理為較少的請求。如需詳細資訊，請參閱日誌管理員元件的[上傳間隔參數](#) (periodicUploadIntervalSec)。

- 日誌格式

您可以選擇 AWS IoT Greengrass Core 軟體是以文字或 JSON 格式寫入日誌。如果您讀取日誌，請選擇文字格式；如果您使用應用程式來讀取或剖析日誌，請選擇 JSON 格式。如需詳細資訊，請參閱 Greengrass 核元件的[日誌格式參數](#) (logging.format)。

- 本機檔案系統日誌資料夾

您可以將日誌資料夾從 `/greengrass/v2/logs` 變更為核心裝置上的另一個資料夾。如需詳細資訊，請參閱 Greengrass 核元件的[輸出目錄參數](#) (logging.outputDirectory)。

AWS CloudTrail 日誌

AWS IoT Greengrass 與整合 AWS CloudTrail，此服務提供使用者、角色或 AWS 服務所採取動作的記錄 AWS IoT Greengrass。如需詳細資訊，請參閱[使用記錄 AWS IoT Greengrass V2 API 呼叫 AWS CloudTrail](#)。

使用 記錄 AWS IoT Greengrass V2 API 呼叫 AWS CloudTrail

AWS IoT Greengrass V2 已與 服務整合 AWS CloudTrail，此服務可提供使用者、角色或 AWS 服務在其中採取之動作的記錄 AWS IoT Greengrass Version 2。CloudTrail 會將 的所有 API 呼叫擷取 AWS IoT Greengrass 為事件。擷取的呼叫包括來自 AWS IoT Greengrass 主控台的呼叫，以及對 AWS IoT Greengrass API 操作的程式碼呼叫。

如果您建立線索，您可以將 CloudTrail 事件持續交付至 S3 儲存貯體，包括 的事件 AWS IoT Greengrass。即使您未設定追蹤，依然可以透過 CloudTrail 主控台的事件歷史記錄檢視最新事件。您可以使用 CloudTrail 所收集的資訊來判斷提出的請求 AWS IoT Greengrass、提出請求的 IP 地址、提出請求的人員、提出請求的時間，以及其他詳細資訊。

如需有關 CloudTrail 的相關資訊，請參閱 [AWS CloudTrail 使用者指南](#)。

主題

- [AWS IoT Greengrass V2 CloudTrail 中的資訊](#)
- [AWS IoT Greengrass CloudTrail 中的資料事件](#)
- [AWS IoT Greengrass CloudTrail 中的管理事件](#)
- [了解 AWS IoT Greengrass V2 日誌檔案項目](#)

AWS IoT Greengrass V2 CloudTrail 中的資訊

建立帳戶 AWS 帳戶 時，您的 上會啟用 CloudTrail。當活動在 中發生時 AWS IoT Greengrass，該活動會與事件歷史記錄中的其他服務 AWS 事件一起記錄在 CloudTrail 事件中。您可以檢視、搜尋和下載 AWS 帳戶的最新事件。如需詳細資訊，請參閱「[使用 CloudTrail 事件歷史記錄檢視事件](#)」。

若要不持續記錄 中的事件 AWS 帳戶，包括 的事件 AWS IoT Greengrass，請建立追蹤。線索可讓 CloudTrail 將日誌檔案傳遞到 S3 儲存貯體。根據預設，當您在主控台中建立線索時，線索會套用至所有 AWS 區域。線索會記錄 AWS 分割區中所有區域的事件，並將日誌檔案傳送到您指定的 S3 儲存貯體。此外，您可以設定其他 AWS 服務，以進一步分析和處理 CloudTrail 日誌中收集的事件資料。如需詳細資訊，請參閱下列內容：

- [建立追蹤的概觀](#)
- [CloudTrail 支援的服務和整合](#)
- [設定 CloudTrail 的 Amazon SNS 通知](#)
- [接收多個區域的 CloudTrail 日誌檔案](#)和[接收多個帳戶的 CloudTrail 日誌檔案](#)

CloudTrail 會記錄所有 AWS IoT Greengrass V2 動作，並記錄在 [AWS IoT Greengrass V2 API 參考](#) 中。例如，對 `CreateComponentVersion`、`CreateDeployment` 和 `CancelDeployment` 動作發出的呼叫會在 CloudTrail 記錄檔案中產生專案。

每一筆事件或日誌專案都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 是否使用根或 AWS Identity and Access Management (IAM) 使用者登入資料提出請求。
- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 請求是否由其他 AWS 服務提出。

如需詳細資訊，請參閱 [CloudTrail userIdentity 元素](#)。

AWS IoT Greengrass CloudTrail 中的資料事件

[資料事件](#) 提供有關在資源上執行或在資源中執行的資源操作的資訊（例如，取得元件版本或部署的組態）。這些也稱為資料平面操作。資料事件通常是大量資料的活動。根據預設，CloudTrail 不會記錄資料事件。CloudTrail 事件歷史記錄不會記錄資料事件。

資料事件需支付額外的費用。如需 CloudTrail 定價的詳細資訊，請參閱 [AWS CloudTrail 定價](#)。

您可以使用 CloudTrail 主控台、AWS CLI 或 CloudTrail API 操作來記錄 AWS IoT Greengrass 資源類型的資料事件。本節中的 [表格](#) 顯示可用的資源類型 AWS IoT Greengrass。

- 若要使用 CloudTrail 主控台記錄資料事件，請建立 [線索](#) 或 [事件資料存放區](#) 以記錄資料事件，或 [更新現有的線索或事件資料存放區](#) 以記錄資料事件。
 1. 選擇資料事件以記錄資料事件。
 2. 從資料事件類型清單中，選擇您要記錄資料事件的資源類型。
 3. 選擇您要使用的日誌選取器範本。您可以記錄資源類型的所有資料事件、記錄所有 `readOnly` 事件、記錄所有 `writeOnly` 事件，或建立自訂日誌選取器範本來篩選 `readOnly`、`eventName` 和 `resources.ARN` 欄位。
- 若要使用記錄資料事件 AWS CLI，請設定 `--advanced-event-selectors` 參數以將 `eventCategory` 欄位設定為 `Data` 並將 `resources.type` 欄位設定為資源類型值（請參閱 [資料表](#)）。您可以新增條件來篩選 `readOnly`、`eventName` 和 `resources.ARN` 欄位的值。
- 若要設定追蹤記錄資料事件，請執行 [put-event-selectors](#) 命令。如需詳細資訊，請參閱 [使用記錄線索的資料事件 AWS CLI](#)。

- 若要設定事件資料存放區來記錄資料事件，請執行 [create-event-data-store](#) 命令來建立新的事件資料存放區來記錄資料事件，或執行 [update-event-data-store](#) 命令來更新現有的事件資料存放區。如需詳細資訊，請參閱 [使用 記錄事件資料存放區的資料事件 AWS CLI](#)。

下表列出 AWS IoT Greengrass 資源類型。資料事件類型 (主控台) 資料行會顯示從 CloudTrail 主控台上的資料事件類型清單中選擇的值。resources.type 值欄會顯示值，您會在使用 AWS CLI 或 CloudTrail APIs 設定進階事件選取器時指定此 resources.type 值。記錄到 CloudTrail 的資料 API 資料行會針對資源類型顯示記錄到 CloudTrail 的 API 呼叫。

資料事件類型 (主控台)	resources.type 值	記錄到 CloudTrail 的資料 API
IoT 憑證	AWS::IoT::Certificate	- VerifyClientDeviceIdentity - VerifyClientDeviceIoTCertificateAssociation
IoT Greengrass 元件版本	AWS::GreengrassV2::ComponentVersion	ResolveComponentCandidates
IoT Greengrass 部署	AWS::GreengrassV2::Deployment	GetDeploymentConfiguration
IoT 物件	AWS::IoT::Thing	- ListThingGroupsForCoreDevices - PutCertificateAuthorities - VerifyClientDeviceIoTCertificateAssociation

您可以設定進階事件選取器來篩選 eventName、readOnly 和 resources.ARN 欄位，以僅記錄對您重要的事件。

在上新增篩選條件 eventName，以包含或排除特定資料 APIs。

如需有關這些欄位的詳細資訊，請參閱 [AdvancedFieldSelector](#)。

下列範例示範如何使用 設定進階選取器 AWS CLI。使用您自己的資訊取代 *TrailName* 和 ##。

Example– 記錄 IoT 物件的資料事件

```
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– 篩選特定 IoT 物件 API

```
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– 記錄所有 Greengrass 資料事件

```
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 CloudTrail 中的管理事件

[管理事件](#) 提供有關在 AWS 帳戶中資源上執行的管理操作的資訊。這些也稱為控制平面操作。根據預設，CloudTrail 記錄管理事件。

AWS IoT Greengrass 會將所有 AWS IoT Greengrass 控制平面操作記錄為管理事件。如需 AWS IoT Greengrass 記錄到 CloudTrail 的 AWS IoT Greengrass 控制平面操作清單，請參閱 [AWS IoT Greengrass API 參考第 2 版](#)。

了解 AWS IoT Greengrass V2 日誌檔案項目

權杖是一種組態，能讓事件以日誌檔案的形式交付至您指定的 S3 儲存貯體。CloudTrail 日誌檔案包含一個或多個日誌項目。事件代表來自任何來源的單一請求。其中包含了請求的動作、動作的日期和時間、請求參數等相關資訊。CloudTrail 日誌檔並非依公有 API 呼叫的堆疊追蹤排序，因此不會以任何特定順序出現。

以下範例顯示的是展示 CreateDeployment 動作的 CloudTrail 日誌項目。

```

{
  "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"

```

```
}
```

從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料

系統運作狀態遙測資料是診斷資料，可協助您監控 Greengrass 核心裝置上關鍵操作的效能。您可以建立專案和應用程式，從邊緣裝置擷取、分析、轉換和報告遙測資料。程序工程師等網域專家可以使用這些應用程式來深入了解機群運作狀態。

您可以使用下列方法來從 Greengrass 核心裝置收集遙測資料：

- 根據預設，Nucleus 遙測發射器元件 - Greengrass 核心裝置上的[核遙測發射器元件](#) (aws.greengrass.telemetry.NucleusEmitter) 會將遙測資料發佈至 \$local/greengrass/telemetry 主題。您可以使用發佈至此主題的資料，在核心裝置上於本機採取行動，即使您的裝置與雲端的連線有限。您也可以選擇性地設定元件，將遙測資料發佈至您選擇的 AWS IoT Core MQTT 主題。

您必須將核發射器元件部署至核心裝置，才能發佈遙測資料。將遙測資料發佈至本機主題不會產生相關費用。不過，使用 MQTT 主題將資料發佈到 AWS 雲端 需要[AWS IoT Core 定價](#)。

AWS IoT Greengrass 提供多種[社群元件](#)，可協助您使用 InfluxDB 和 Grafana 在本機核心裝置上分析和視覺化遙測資料。這些元件使用來自 核發射器元件的遙測資料。如需詳細資訊，請參閱 [InfluxDB 發佈者元件](#) 的 README。

- 遙測代理程式 - Greengrass 核心裝置上的遙測代理程式會收集本機遙測資料，並將其發佈至 Amazon EventBridge，而不需要任何客戶互動。核心裝置會盡最大努力將遙測資料發佈至 EventBridge。例如，核心裝置可能無法在離線時交付遙測資料。

預設會為所有 Greengrass 核心裝置啟用遙測代理程式功能。一旦您設定 Greengrass 核心裝置，就會自動開始接收資料。除了您的資料連結成本之外，從核心裝置到的資料傳輸 AWS IoT Core 是免費的。這是因為代理程式發佈到 AWS 預留主題。不過，視您的使用案例而定，您可能會在接收或處理資料時產生成本。

Note

Amazon EventBridge 是一種事件匯流排服務，可用來將應用程式與各種來源的資料連線，例如 Greengrass 核心裝置。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[什麼是 Amazon EventBridge？](#)

為了確保 AWS IoT Greengrass Core 軟體正常運作，AWS IoT Greengrass 會將資料用於開發和品質改善目的。此功能也有助於通知新的和增強的邊緣功能。最多可 AWS IoT Greengrass 保留七天的遙測資料。

本節說明如何設定和使用遙測代理程式。如需設定核遙測發射器元件的資訊，請參閱 [Nucleus 遙測發射器](#)。

主題

- [遙測指標](#)
- [設定遙測代理程式設定](#)
- [訂閱 EventBridge 中的遙測資料](#)

遙測指標

下表說明遙測代理程式發佈的指標。

名稱	描述	
系統		
SystemMemUsage	Greengrass 核心裝置上所有應用程式目前使用的記憶體數量，包括作業系統。	
CpuUsage	Greengrass 核心裝置上所有應用程式目前使用的 CPU 數量，包括作業系統。	
TotalNumberOfFDs	Greengrass 核心裝置的作業系統存放的檔案描述項數量。一個檔案描述項可唯一識別一個開啟的檔案。	
Greengrass 核		
NumberOfComponentsRunning	在 Greengrass 核心裝置上執行的元件數量。	

名稱	描述	
NumberOfComponentsErrored	Greengrass 核心裝置上處於錯誤狀態的元件數量。	
NumberOfComponentsInstalled	安裝在 Greengrass 核心裝置上的元件數量。	
NumberOfComponentsStarting	在 Greengrass 核心裝置上啟動的元件數量。	
NumberOfComponentsNew	Greengrass 核心裝置上的新元件數量。	
NumberOfComponentsStopping	Greengrass 核心裝置上停止的元件數量。	
NumberOfComponentsFinished	Greengrass 核心裝置上完成的元件數量。	
NumberOfComponentsBroken	Greengrass 核心裝置上損壞的元件數量。	
NumberOfComponentsStateless	Greengrass 核心裝置上無狀態的元件數量。	
用戶端裝置驗證 – 此功能需要 v2.4.0 或更新版本的用戶端裝置驗證元件。		
VerifyClientDeviceIdentity.Success	驗證用戶端裝置身分是否成功的次數。	
VerifyClientDeviceIdentity.Failure	驗證用戶端裝置身分失敗的次數。	
AuthorizeClientDeviceActions.Success	用戶端裝置獲授權完成請求動作的次數。	

名稱	描述	
AuthorizeClientDeviceActions.Failure	用戶端裝置無權完成請求動作的次數。	
GetClientDeviceAuthToken.Success	用戶端裝置成功驗證的次數。	
GetClientDeviceAuthToken.Failure	用戶端裝置無法驗證的次數。	
SubscribeToCertificateUpdates.Success	憑證更新的成功訂閱數量。	
SubscribeToCertificateUpdates.Failure	嘗試訂閱憑證更新的失敗次數。	
ServiceError	用戶端裝置驗證中未處理的內部錯誤數目。	
串流管理員 – 此功能需要 2.7.0 版或更新版本的 Greengrass 核心元件。		
BytesAppended	附加至串流管理員的資料位元組數。	
BytesUploadedToIoTAnalytics	串流管理員匯出至頻道的資料位元組數 AWS IoT Analytics。	
BytesUploadedToKinesis	串流管理員匯出至 Amazon Kinesis Data Streams 中串流的資料位元組數。	
BytesUploadedToIoTSiteWise	串流管理員匯出至資產屬性的資料位元組數 AWS IoT SiteWise。	

名稱	描述	
BytesUploadedToS3	串流管理員匯出至 Amazon S3 物件的資料位元組數。	
系統指標 – 此功能需要 2.15.0 版或更新版本的 Greengrass 核心元件。		
CPUArchitecture	裝置中央處理單元架構。	
Family	裝置作業系統系列 (僅限 Windows)。	
KernelVersion	裝置核心版本 (僅限 Unix)。	
KnowledgeBaseArticles	安裝在裝置上的知識庫文章 (僅限 Windows)。	
OSBuildMajor	作業系統版本的主要建置編號 (僅限 Windows)。	
OSBuildMinor	作業系統版本的次要建置編號 (僅限 Windows)。	
OSName	裝置作業系統的名稱。	
OSVersion	裝置作業系統的行銷版本。	

設定遙測代理程式設定

遙測代理程式使用以下預設設定：

- 遙測代理程式每小時彙總一次遙測資料。
- 遙測代理程式每 24 小時發佈一次遙測訊息。

遙測代理程式會使用服務品質 (QoS) 層級為 0 的 MQTT 通訊協定來發佈資料，這表示它不會確認交付或重試發佈嘗試。遙測訊息與目標為之訂閱的其他訊息共用 MQTT 連線 AWS IoT Core。

除了您的資料連結成本之外，從核心到 的資料傳輸 AWS IoT Core 是免費的。這是因為代理程式發佈到 AWS 預留主題。不過，視您的使用案例而定，您可能會在接收或處理資料時產生成本。

您可以啟用或停用每個 Greengrass 核心裝置的遙測代理程式功能。您也可以設定核心裝置彙總和發佈資料的間隔。若要設定遙測，請在部署 [Greengrass 核元件](#) 時自訂 [遙測組態參數](#)。

訂閱 EventBridge 中的遙測資料

您可以在 Amazon EventBridge 中建立規則，定義如何處理從 Greengrass 核心裝置上的遙測代理程式發佈的遙測資料。EventBridge 收到資料時，會叫用規則中定義的目標動作。例如，您可以建立事件規則來傳送通知、存放事件資訊、採取修正動作或叫用其他事件。

遙測事件

遙測事件使用以下格式。

```
{
  "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)"
    }
  ],
]
}
]
}
}

```

ADP 陣列包含具有下列屬性的彙總資料點清單：

TS

收集資料時的時間戳記。

NS

指標命名空間。

M

指標清單。指標包含下列屬性：

N

指標的名稱

Sum

此遙測事件中指標值的總和。

U

指標值的單位。

如需每個指標的詳細資訊，請參閱[遙測指標](#)。

建立 EventBridge 規則的先決條件

為建立 EventBridge 規則之前 AWS IoT Greengrass，您應該執行下列動作：

- 熟悉 Eventbridge 中的事件、規則和目標。
- 建立和設定 EventBridge 規則調用的[目標](#)。規則可以叫用許多類型的目標，例如 Amazon Kinesis 串流、AWS Lambda 函數、Amazon SNS 主題和 Amazon SQS 佇列。

您的 EventBridge 規則和相關聯的目標必須位於您建立 Greengrass 資源 AWS 區域的 中。如需詳細資訊，請參閱《AWS 一般參考》中的[服務端點和配額](#)。

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[什麼是 Amazon EventBridge？](#) 和 [Amazon EventBridge 入門](#)。

建立事件規則以取得遙測資料（主控台）

使用下列步驟來使用 AWS 管理主控台 建立 EventBridge 規則，以接收 Greengrass 核心裝置發佈的遙測資料。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。如需詳細資訊，請參閱《Amazon [EventBridge 使用者指南](#)》中的從 [AWS 資源建立事件觸發的](#) EventBridge 規則。 EventBridge

1. 開啟 [Amazon EventBridge 主控台](#)，然後選擇建立規則。

2. 在 Name and description (名稱和描述)，輸入規則的名稱和描述。
3. 在 Define pattern (定義模式)，設定規則模式。
 - a. 選擇 Event pattern (事件模式)。
 - b. 選擇 Pre-defined pattern by service (依服務預先定義模式)。
 - c. 在 Service provider (服務提供者)，選擇 AWS。
 - d. 在 Service Name (服務名稱)，選擇 Greengrass。
 - e. 針對事件類型，選取 Greengrass 遙測資料。
4. 在 Select event bus (選取事件匯流排)，保留預設的事件匯流排選項。
5. 在 Select targets (選取目標)，設定您的目標。下列範例使用 Amazon SQS 佇列，但您可以設定其他目標類型。
 - a. 針對目標，選擇 SQS 佇列。
 - b. 針對佇列*，選擇您的目標佇列。
6. 在 Tags - optional (標籤 - 選用)，定義規則的標籤，或將欄位留白。
7. 選擇建立。

建立事件規則以取得遙測資料 (CLI)

使用下列步驟來使用 AWS CLI 建立 EventBridge 規則，以接收 Greengrass 核心裝置發佈的遙測資料。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。

1. 建立 規則。
 - 以核心裝置的物件名稱取代 *thing-name*。

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\"]}}"
```

模式省略的屬性會遭到忽略。

2. 新增主題作為規則目標。下列範例使用 Amazon SQS，但您可以設定其他目標類型。

- 將 *queue-arn* 取代為 Amazon SQS 佇列的 ARN。

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

若要允許 Amazon EventBridge 叫用您的目標佇列，您必須將資源型政策新增至主題。如需詳細資訊，請參閱 [《Amazon EventBridge 使用者指南》](#) 中的 [Amazon SQS 許可](#)。EventBridge

如需詳細資訊，請參閱 [《Amazon EventBridge 使用者指南》](#) 中的 [EventBridge 中的事件和事件模式](#)。

取得部署和元件運作狀態通知

Amazon EventBridge 事件規則會針對裝置收到的 Greengrass 部署和裝置上已安裝的元件，提供您狀態變更的通知。EventBridge 提供近乎即時的系統事件串流，描述 AWS 資源的變更。會盡最大努力將這些事件 AWS IoT Greengrass 傳送至 EventBridge。這表示 會 AWS IoT Greengrass 嘗試將所有事件傳送至 EventBridge，但在某些情況下，可能不會傳送事件。此外，AWS IoT Greengrass 可能會傳送指定事件的多個副本，這表示您的事件接聽程式可能不會按照事件發生的順序接收事件。

Note

Amazon EventBridge 是一種事件匯流排服務，可用來將應用程式與各種來源的資料連線，例如 [Greengrass 核心裝置](#)、部署和元件通知。如需詳細資訊，請參閱 [《Amazon EventBridge 使用者指南》](#) 中的 [什麼是 Amazon EventBridge？](#)

主題

- [部署狀態變更事件](#)
- [元件狀態變更事件](#)
- [建立 EventBridge 規則的先決條件](#)
- [設定裝置運作狀態通知（主控台）](#)
- [設定裝置運作狀態通知 \(CLI\)](#)
- [設定裝置運作狀態通知 \(CloudFormation\)](#)
- [另請參閱](#)

部署狀態變更事件

AWS IoT Greengrass 當部署進入下列狀態時發出事件：FAILED、SUCCEEDEDCOMPLETED、、REJECTED和 CANCELED。您可以建立 EventBridge 規則，針對所有狀態轉換或轉換至您指定的狀態執行該規則。當部署進入啟動規則的狀態時，EventBridge 會叫用規則中定義的目標動作。這可讓您傳送通知、擷取事件資訊、採取修正動作，或啟動其他事件來回應狀態變更。例如，您可以為下列使用案例建立規則：

- 啟動部署後操作，例如下載資產和通知人員。
- 部署成功或失敗後傳送通知。
- 發佈關於部署事件的自訂指標。

部署狀態變更的[事件](#)會使用下列格式：

```
{
  "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/
1uLCYANGq08RIH+x2H+hEKc=)"
  }
}
```

```
}
```

您可以建立規則和事件，以更新部署狀態。當部署完成 FAILED、SUCCEEDED、REJECTED、或時 COMPLETED，會啟動事件 CANCELED。如果核心裝置上的部署失敗，您將收到詳細的回應，說明部署失敗的原因。如需部署錯誤代碼的詳細資訊，請參閱 [詳細部署錯誤代碼](#)。

部署狀態

- FAILED。部署失敗。
- SUCCEEDED。針對物件群組的部署已成功完成。
- COMPLETED。目標為物件的部署已成功完成。
- REJECTED。部署遭拒。如需詳細資訊，請參閱 statusDetails 欄位。
- CANCELED。使用者已取消部署。

事件可能會重複或不按順序。若要判斷事件的順序，請使用 time 屬性。

如需 errorStacks 和 中錯誤代碼的完整清單 errorTypes，請參閱 [詳細部署錯誤代碼](#) 和 [詳細的元件狀態碼](#)。

元件狀態變更事件

對於 2.12.2 版和更早 AWS IoT Greengrass 版本，Greengrass 會在元件進入下列狀態時發出事件：ERRORED 和 BROKEN。對於 Greengrass 核版本 2.12.3 和更新版本，Greengrass 會在元件進入下列狀態時發出事件：ERRORED、RUNNING、BROKEN 和 FINISHED。當部署完成時，Greengrass 也會發出事件。您可以建立 EventBridge 規則，針對所有狀態轉換或轉換至您指定的狀態執行該規則。當已安裝的元件進入啟動規則的狀態時，EventBridge 會叫用規則中定義的目標動作。這可讓您傳送通知、擷取事件資訊、採取修正動作，或啟動其他事件來回應狀態變更。

元件狀態變更 [的事件](#) 使用以下格式：

Greengrass nucleus v2.12.2 and earlier

<title>元件狀態：ERRORED 或 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>元件狀態： ERRORED 或 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>元件狀態：RUNNING 或 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
      }
    ]
  }
}
```

您可以建立規則和事件，以更新已安裝元件的狀態。當元件變更裝置上的狀態時，會啟動事件。您將會收到詳細的回應，說明元件發生錯誤或損壞的原因。您也會收到狀態碼，指出失敗的原因。如需元件狀態碼的詳細資訊，請參閱 [詳細的元件狀態碼](#)。

建立 EventBridge 規則的先決條件

為建立 EventBridge 規則之前 AWS IoT Greengrass，請執行下列動作：

- 熟悉 Eventbridge 中的事件、規則和目標。
- 建立和設定將由您的 Eventbridge 規則叫用的目標。規則可以叫用許多類型的目標，包括：
 - Amazon Simple Notification Service (Amazon SNS)

- AWS Lambda 函數
- Amazon Kinesis Video Streams
- Amazon Simple Queue Service (Amazon SQS) 佇列

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[什麼是 Amazon EventBridge？](#) 和 [Amazon EventBridge 入門](#)。

設定裝置運作狀態通知（主控台）

使用下列步驟建立 EventBridge 規則，在群組的部署狀態變更時發佈 Amazon SNS 主題。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的[從 AWS 資源建立事件觸發的 EventBridge 規則](#)。 EventBridge

1. 開啟 [Amazon EventBridge 主控台](#)。
2. 在導覽窗格中，選擇規則。
3. 選擇建立規則。
4. 輸入規則的名稱和描述。

在同一個區域和同一個事件匯流排上，規則不能與另一個規則同名。

5. 針對事件匯流排，選擇要與此規則建立關聯的事件匯流排。如果您想要此規則匹配來自您的帳戶的事件，請選取 AWS 預設事件匯流排。當您帳戶中 AWS 的服務發出事件時，一律會前往您帳戶的預設事件匯流排。
6. 針對規則類型，選擇具有事件模式的規則。
7. 選擇下一步。
8. 在事件來源，選擇 AWS 事件。
9. 針對事件模式，選擇 AWS 服務。
10. 針對AWS 服務，選擇 Greengrass。
11. 針對事件類型，從下列選項中選擇：
 - 針對部署事件，選擇 Greengrass V2 有效部署狀態變更。
 - 針對元件事件，選擇 Greengrass V2 已安裝元件狀態變更。
12. 選擇下一步。
13. 在目標類型欄位中，選擇 AWS 服務。

14. 針對選取目標，設定您的目標。此範例使用 Amazon SNS 主題，但您可以設定其他目標類型來傳送通知。
 - a. 在 Target (目標)，選擇 SNS topic (SNS 主題)。
 - b. 在 Topic (主題)，選擇您的目標主題。
 - c. 選擇下一步。
15. 選擇下一步。
16. 檢閱規則的詳細資訊，然後選擇建立規則。

設定裝置運作狀態通知 (CLI)

使用下列步驟建立 EventBridge 規則，在發生 Greengrass 狀態變更事件時發佈 Amazon SNS 主題。藉由這個規則，Web 伺服器、電子郵件地址和其他主題訂閱者將能回應事件。

1. 建立 規則。
 - 對於部署狀態變更事件。

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":  
  [\"Greengrass V2 Effective Deployment Status Change\"]}"
```

- 對於元件狀態變更事件。

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":  
  [\"Greengrass V2 Installed Component Status Change\"]}"
```

模式省略的屬性會遭到忽略。

2. 新增主題作為規則目標。
 - 將 *topic-arn* 取代為 Amazon SNS 主題的 ARN。

```
aws events put-targets \  
  --rule TestRule \  
  --targets [{"arn": "arn:aws:sns:us-east-1:123456789012:my-topic"}]
```

```
--targets "Id"="1", "Arn"="topic-arn"
```

Note

若要讓 Amazon EventBridge 呼叫您的目標主題，您必須將以資源為基礎的政策新增到您的主題。如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的 [Amazon SNS 許可](#)。

如需詳細資訊，請參閱《Amazon EventBridge 使用者指南》中的 [EventBridge 中的事件和事件模式](#)。

設定裝置運作狀態通知 (CloudFormation)

使用 CloudFormation 範本建立 EventBridge 規則，傳送 Greengrass 群組部署狀態變更的通知。如需詳細資訊，請參閱《AWS CloudFormation 使用者指南》中的 [Amazon EventBridge 資源類型參考](#)。

另請參閱

- [檢查裝置部署狀態](#)
- 《[Amazon EventBridge 使用者指南](#)》中的什麼是 [Amazon EventBridge](#)？ EventBridge

檢查 Greengrass 核心裝置狀態

Greengrass 核心裝置報告其軟體元件的狀態 AWS IoT Greengrass。您可以檢查每個裝置的運作狀態摘要，也可以檢查每個裝置上每個元件的狀態。

核心裝置的運作狀態如下：

- HEALTHY – AWS IoT Greengrass 核心軟體和所有元件在核心裝置上執行時不會發生問題。
- UNHEALTHY – AWS IoT Greengrass 核心軟體或元件在核心裝置上處於錯誤狀態。

Note

AWS IoT Greengrass 倚賴個別裝置傳送狀態更新至 AWS 雲端。如果 AWS IoT Greengrass Core 軟體未在裝置上執行，或裝置未連線至 AWS 雲端，則該裝置回報的狀態可能不會反映其目前狀態。狀態時間戳記指出裝置狀態上次更新的時間。
核心裝置會在下列時間傳送狀態更新：

- AWS IoT Greengrass 核心軟體啟動時
- 當核心裝置從 接收部署時 AWS 雲端
- 對於 Greengrass 核 2.12.2 及更早版本，核心裝置會在核心裝置上任何元件的狀態變成 ERRORED或 BROKEN
- 對於 Greengrass 核 2.12.3 和更新版本，核心裝置會在核心裝置上任何元件的狀態變成 ERRORED、RUNNING、BROKEN或 時傳送狀態更新 FINISHED
- 您可以[設定的定期間隔](#)，預設為 24 小時

對於 AWS IoT Greengrass Core v2.7.0 和更新版本，核心裝置會在本機部署和雲端部署發生時傳送狀態更新

主題

- [檢查核心裝置的運作狀態](#)
- [檢查核心裝置群組的運作狀態](#)
- [檢查核心裝置元件狀態](#)

檢查核心裝置的運作狀態

您可以檢查個別核心裝置的狀態。

檢查核心裝置的狀態 (AWS CLI)

- 執行下列命令來擷取裝置的狀態。將 *coreDeviceName* 取代為要查詢的核心裝置名稱。

```
aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
```

回應包含核心裝置的相關資訊，包括其狀態。

檢查核心裝置群組的運作狀態

您可以檢查核心裝置群組（物件群組）的狀態。

檢查裝置群組的狀態 (AWS CLI)

- 執行下列命令來擷取多個核心裝置的狀態。將命令中的 ARN 取代為要查詢之物件群組的 ARN。

```
aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
```

回應包含物件群組中核心裝置的清單。清單中的每個項目都包含核心裝置的狀態。

檢查核心裝置元件狀態

您可以在核心裝置上檢查軟體元件的狀態，例如生命週期狀態。如需元件生命週期狀態的詳細資訊，請參閱[開發 AWS IoT Greengrass 元件](#)。

若要檢查核心裝置上的元件狀態 (AWS CLI)

- 執行下列命令來擷取核心裝置上的元件狀態。將 *coreDeviceName* 取代為要查詢的核心裝置名稱。

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

回應包含在核心裝置上執行的元件清單。清單中的每個項目都包含元件的生命週期狀態，包括資料的目前狀態，以及 Greengrass 核心裝置上次將包含特定元件的訊息傳送到雲端的時間。回應也會包含將元件帶入 Greengrass 核心裝置的最新部署來源。

Note

此命令會擷取 Greengrass 核心裝置執行的元件分頁清單。根據預設，此清單不包含部署為其他元件相依性的元件。您可以將 `topologyFilter` 參數設定為 `ALL`，在回應中包含相依性 ALL。

執行 AWS Lambda 函數

Note

AWS IoT Greengrass 目前不支援 Windows 核心裝置上的此功能。

您可以將 AWS Lambda 函數匯入為在 AWS IoT Greengrass 核心裝置上執行的元件。在下列情況中，您可能想要執行此操作：

- 您在 Lambda 函數中具有應用程式碼，而您要部署到核心裝置。
- 您有要在 AWS IoT Greengrass V2 核心裝置上執行的 AWS IoT Greengrass V1 應用程式。如需詳細資訊，請參閱[步驟 2：建立和部署 AWS IoT Greengrass V2 元件以遷移 AWS IoT Greengrass V1 應用程式](#)。

Lambda 函數包含下列元件的相依性。匯入函數時，您不需要將這些元件定義為相依性。當您部署 Lambda 函數元件時，部署會包含這些 Lambda 元件相依性。

- [Lambda 啟動器元件](#) (`aws.greengrass.LambdaLauncher`) 會處理程序和環境組態。
- [Lambda 管理員元件](#) (`aws.greengrass.LambdaManager`) 會處理程序間通訊和擴展。
- [Lambda 執行期元件](#) (`aws.greengrass.LambdaRuntimes`) 為每個支援的 Lambda 執行期提供成品。

主題

- [要求](#)
- [設定 Lambda 函數生命週期](#)
- [設定 Lambda 函數容器化](#)
- [將 Lambda 函數匯入為元件（主控台）](#)
- [將 Lambda 函數匯入為元件 \(AWS CLI\)](#)

要求

您的核心裝置和 Lambda 函數必須符合下列要求，才能在 AWS IoT Greengrass Core 軟體上執行函數：

- 您的核心裝置必須符合執行 Lambda 函數的要求。如果您想要核心裝置執行容器化 Lambda 函數，裝置必須符合執行此作業的要求。如需詳細資訊，請參閱[Lambda 函數需求](#)。
- 您必須安裝 Lambda 函數在核心裝置上使用的程式設計語言。

Tip

您可以建立安裝程式設計語言的元件，然後將該元件指定為 Lambda 函數元件的相依性。Greengrass 支援所有 Lambda 支援的 Python、Node.js 和 Java 執行時間版本。Greengrass 不會對已取代的 Lambda 執行時間版本套用任何其他限制。您可以在上執行使用這些已棄用執行時間的 Lambda 函數 AWS IoT Greengrass，但無法在中建立它們 AWS Lambda。如需 AWS IoT Greengrass 支援 Lambda 執行時間的詳細資訊，請參閱[執行 AWS Lambda 函數](#)。

設定 Lambda 函數生命週期

Greengrass Lambda 函數生命週期會決定函數何時啟動，以及它如何建立和使用容器。生命週期也會決定 AWS IoT Greengrass Core 軟體如何保留函數處理常式外部的變數和預先處理邏輯。

AWS IoT Greengrass 支援隨需（預設）和長期生命週期：

- 隨需函數會在叫用時啟動，並在沒有任務剩餘時停止。除非現有容器可供重複使用，否則函數的每次調用都會建立個別容器，也稱為沙盒，以處理調用。任何容器都可能處理您傳送至函數的資料。

隨需函數的多個調用可以同時執行。

建立新容器時，不會保留您在函數處理常式外定義的變數和預先處理邏輯。

- 長期（或固定）函數會在 AWS IoT Greengrass Core 軟體在單一容器中啟動和執行時啟動。相同的容器會處理您傳送至函數的所有資料。

多個調用會排入佇列，直到 AWS IoT Greengrass 核心軟體提早執行調用。

您於函數處理常式外部定義的變數和預先處理邏輯，會在每次呼叫處理常式時保留。

當您需要在沒有任何初始輸入的情況下開始工作時，請使用長期 Lambda 函數。例如，長期函數可以載入並開始處理機器學習模型，以便在函數接收裝置資料時準備就緒。

Note

長期函數具有與其處理常式的每個調用相關聯的逾時。如果您想要叫用無限期執行的程式碼，您必須在處理常式外部啟動它。請確定處理常式外部沒有封鎖碼，而可能使函數無法初始化。

除非 AWS IoT Greengrass 核心軟體停止，否則這些函數會執行，例如在部署或重新啟動期間。如果函數遇到未攔截的例外狀況、超過其記憶體限制，或輸入錯誤狀態，例如處理常式逾時，則不會執行這些函數。

如需容器重複使用的詳細資訊，請參閱AWS 運算部落格中的[了解 中的容器重複使用 AWS Lambda](#)。

設定 Lambda 函數容器化

根據預設，Lambda 函數會在 AWS IoT Greengrass 容器內執行。Greengrass 容器可在函數和主機之間提供隔離。此隔離可提高主機和容器中函數的安全性。

建議您在 Greengrass 容器中執行 Lambda 函數，除非您的使用案例需要它們在沒有容器化的情況下執行。透過在 Greengrass 容器中執行 Lambda 函數，您可以更好地控制如何限制對資源的存取。

在下列情況下，您可以執行 Lambda 函數而不進行容器化：

- 您想要 AWS IoT Greengrass 在不支援容器模式的裝置上執行。例如，如果您想要使用特殊的 Linux 發行版本，或是舊版的核心版本已過期。
- 您想要在另一個容器環境中使用自己的 OverlayFS 執行 Lambda 函數，但當您在 Greengrass 容器中執行時，會遇到 OverlayFS 衝突。
- 您需要使用無法在部署時間確定的路徑，或其路徑在部署後可能會變更的路徑來存取本機資源。此資源的範例為可插入裝置。
- 您有一個先前以程序形式寫入的應用程式，當您在 Greengrass 容器中執行時遇到問題。

容器化差異

容器化	備註
Greengrass 容器	• 當您在 Greengrass 容器中執行 Lambda 函數時，所有 AWS IoT Greengrass 功能都可使用。

容器化	備註
	- 在 Greengrass 容器中執行的 Lambda 函數無法存取其他 Lambda 函數的部署程式碼，即使它們使用相同的系統群組執行。換句話說，您的 Lambda 函數會執行時彼此的隔離增加。 - 由於 AWS IoT Greengrass Core 軟體會在與 Lambda 函數相同的容器中執行所有子程序，因此子程序會在 Lambda 函數停止時停止。
沒有容器	- 下列功能不適用於非容器化 Lambda 函數： - Lambda 函數記憶體限制。 - 本機裝置和磁碟區資源。您必須使用核心裝置上的檔案路徑來存取這些資源，而不是做為 Lambda 函數資源。 - 如果您的非容器化 Lambda 函數存取機器學習資源，您必須識別資源擁有者，並在資源上設定存取許可，而不是在 Lambda 函數上。 - 非容器化 Lambda 函數對使用相同系統群組執行的其他 Lambda 函數的部署程式碼具有唯讀存取權。

如果您在部署 Lambda 函數時變更其容器化，該函數可能無法如預期般運作。如果 Lambda 函數使用不再可用於新容器化設定的本機資源，則部署會失敗。

- 當您將 Lambda 函數從在 Greengrass 容器中執行變更為在沒有容器化的情況下執行時，函數的記憶體限制會遭到捨棄。您必須直接存取檔案系統，而不是使用連接的本機資源。部署 Lambda 函數之前，您必須先移除任何連接的資源。
- 當您將 Lambda 函數從在沒有容器化的情況下執行變更為在容器中執行時，您的 Lambda 函數會失去檔案系統的直接存取權。您必須為每個函數定義記憶體限制，或接受預設的 16 MB 記憶體限制。您可以在部署時為每個 Lambda 函數設定這些設定。

若要變更 Lambda 函數元件的容器化設定，請在部署元件時，將 `containerMode` 組態參數的值設定為下列其中一個選項。

- `NoContainer` – 元件不會在隔離的執行時間環境中執行。

- GreengrassContainer – 元件會在 AWS IoT Greengrass 容器內的隔離執行期環境中執行。

如需如何部署和設定元件的詳細資訊，請參閱 [將 AWS IoT Greengrass 元件部署至裝置](#) 和 [更新元件組態](#)。

將 Lambda 函數匯入為元件（主控台）

當您使用 [AWS IoT Greengrass 主控台](#) 建立 Lambda 函數元件時，您可以匯入現有的 AWS Lambda 函數，然後將其設定為建立在 Greengrass 裝置上執行的元件。

開始之前，請檢閱在 Greengrass 裝置上執行 Lambda 函數 [的需求](#)。

任務

- [步驟 1：選擇要匯入的 Lambda 函數](#)
- [步驟 2：設定 Lambda 函數參數](#)
- [步驟 3：（選用）指定 Lambda 函數支援的平台](#)
- [步驟 4：（選用）指定 Lambda 函數的元件相依性](#)
- [步驟 5：（選用）在容器中執行 Lambda 函數](#)
- [步驟 6：建立 Lambda 函數元件](#)

步驟 1：選擇要匯入的 Lambda 函數

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇元件。
2. 在元件頁面上，選擇建立元件。
3. 在建立元件頁面的元件資訊下，選擇匯入 Lambda 函數。
4. 在 Lambda 函數中，搜尋並選擇您要匯入的 Lambda 函數。

AWS IoT Greengrass 會使用 Lambda 函數的名稱建立元件。

5. 在 Lambda 函數版本中，選擇要匯入的版本。您無法選擇 Lambda 別名，例如 \$LATEST。

AWS IoT Greengrass 會使用 Lambda 函數的版本建立元件，做為有效的語意版本。例如，如果您的函數版本為 3，則元件版本會變成 3.0.0。

步驟 2：設定 Lambda 函數參數

在建立元件頁面的 Lambda 函數組態下，設定下列參數以用於執行 Lambda 函數。

1. （選用）新增 Lambda 函數訂閱工作訊息的事件來源清單。您可以指定事件來源，將此函數訂閱本機發佈/訂閱訊息和 AWS IoT Core MQTT 訊息。從事件來源接收訊息時，會呼叫 Lambda 函數。

Note

若要將此函數訂閱其他 Lambda 函數或元件的訊息，請在部署此 Lambda 函數元件時部署 [舊版訂閱路由器](#) 元件。部署舊版訂閱路由器元件時，請指定 Lambda 函數使用的訂閱。

在事件來源下，執行下列動作來新增事件來源：

- a. 針對您新增的每個事件來源，指定下列選項：

- 主題 – 訂閱訊息的主題。
- 類型 – 事件來源的類型。您可以從以下選項中選擇：
 - 本機發佈/訂閱 – 訂閱本機發佈/訂閱訊息。

如果您使用 [Greengrass 核 v2.6.0](#) 或更新版本，以及 [Lambda Manager v2.2.5](#) 或更新版本，則可以在指定此類型時使用主題中的 MQTT 主題萬用字元 (+ 和 #)。

- AWS IoT Core MQTT – 訂閱 AWS IoT Core MQTT 訊息。

當您指定此類型時，您可以在主題中使用 MQTT 主題萬用字元 (+ 和 #)。

- b. 若要新增另一個事件來源，請選擇新增事件來源並重複上一個步驟。若要移除事件來源，請選擇您要移除的事件來源旁的移除。
2. 針對逾時（秒），輸入非固定 Lambda 函數在逾時之前可執行的時間上限，以秒為單位。預設為 3 秒。
 3. 針對固定，選擇是否固定 Lambda 函數元件。預設值為 True。
 - 固定（或長期）Lambda 函數會在 AWS IoT Greengrass 啟動時啟動，並在自己的容器中持續執行。
 - 非固定（或隨需）Lambda 函數只會在接收工作項目時啟動，並在閒置達指定的閒置時間上限後結束。如果函數有多個工作項目，則 AWS IoT Greengrass Core 軟體會建立函數的多個執行個體。

4. (選用) 在其他參數下，設定下列 Lambda 函數參數。

- 狀態逾時 (秒) – Lambda 函數元件將狀態更新傳送至 Lambda 管理員元件的間隔，以秒為單位。此參數僅適用於固定函數。預設值為 60 秒。
- 最大佇列大小 – Lambda 函數元件的訊息佇列大小上限。Core AWS IoT Greengrass 軟體會將訊息存放在 FIFO (先進先出) 佇列中，直到可以執行 Lambda 函數來取用每個訊息為止。預設值為 1,000 則訊息。
- 執行個體數目上限 – 非固定 Lambda 函數可同時執行的執行個體數目上限。預設值為 100 個執行個體。
- 閒置時間上限 (秒) – 在 AWS IoT Greengrass 核心軟體停止其程序之前，非固定 Lambda 函數可以閒置的時間上限，以秒為單位。預設值為 60 秒。
- 編碼類型 – Lambda 函數支援的承載類型。您可以從以下選項中選擇：
 - JSON
 - 二進位

預設值為 JSON。

5. (選用) 指定命令列引數清單，以在執行時傳遞至 Lambda 函數。

- a. 在其他參數、程序引數下，選擇新增引數。
- b. 針對您新增的每個引數，輸入您要傳遞至函數的引數。
- c. 若要移除引數，請選擇您要移除的引數旁的移除。

6. (選用) 指定 Lambda 函數在執行時可用的環境變數。環境變數可讓您儲存和更新組態設定，而不需要變更函數程式碼。

- a. 在其他參數、環境變數下，選擇新增環境變數。
- b. 針對您新增的每個環境變數，指定下列選項：
 - 索引鍵 – 變數名稱。
 - 值 – 此變數的預設值。
- c. 若要移除環境變數，請選擇您要移除的環境變數旁的移除。

步驟 3：（選用）指定 Lambda 函數支援的平台

所有核心裝置都有作業系統和架構的屬性。當您部署 Lambda 函數元件時，AWS IoT Greengrass Core 軟體會將您指定的平台值與核心裝置上的平台屬性進行比較，以判斷該裝置上是否支援 Lambda 函數。

Note

您也可以將 Greengrass 核元件部署至核心裝置時指定自訂平台屬性。如需詳細資訊，請參閱 [平台覆寫 Greengrass 核元件的參數](#)。 ???

在 Lambda 函數組態下，其他參數、平台會執行下列動作，以指定此 Lambda 函數支援的平台。

1. 針對每個平台，指定下列選項：
 - 作業系統 – 平台的作業系統名稱。目前，僅支援的值為 linux。
 - 架構 – 平台的處理器架構。支援的值如下：
 - amd64
 - arm
 - aarch64
 - x86
2. 若要新增另一個平台，請選擇新增平台並重複上一個步驟。若要移除支援的平台，請選擇您要移除之平台旁的移除。

步驟 4：（選用）指定 Lambda 函數的元件相依性

元件相依性會識別函數使用的其他 AWS 提供的元件或自訂元件。當您部署 Lambda 函數元件時，部署會包含要讓函數執行的這些相依性。

Important

若要匯入您建立以在 AWS IoT Greengrass V1 上執行的 Lambda 函數，您必須為函數使用的功能定義個別元件相依性，例如秘密、本機陰影和串流管理員。將這些元件定義為 [硬相依性](#)，以便在相依性變更狀態時重新啟動 Lambda 函數元件。如需詳細資訊，請參閱 [匯入 V1 Lambda 函數](#)。

在 Lambda 函數組態、其他參數、元件相依性下，完成下列步驟以指定 Lambda 函數的元件相依性。

1. 選擇新增相依性。
2. 針對您新增的每個元件相依性，指定下列選項：
 - 元件名稱 – 元件名稱。例如，輸入 `aws.greengrass.StreamManager` 以包含 [串流管理員元件](#)。
 - 版本需求 – 識別此元件相依性的相容版本的 npm 樣式語意版本限制。您可以指定單一版本或一系列版本。例如，輸入 `^1.0.0` 指定此 Lambda 函數取決於串流管理員元件第一個主要版本中的任何版本。如需語意版本限制的詳細資訊，請參閱 [npm 轉換器計算器](#)。
 - 類型 — 相依性的類型。您可以從以下選項中選擇：
 - 硬 – 如果相依性變更狀態，Lambda 函數元件會重新啟動。這是預設選擇。
 - 軟 – 如果相依性變更狀態，Lambda 函數元件不會重新啟動。
3. 若要移除元件相依性，請選擇元件相依性旁的移除

步驟 5：（選用）在容器中執行 Lambda 函數

根據預設，Lambda 函數會在 AWS IoT Greengrass Core 軟體內的隔離執行期環境中執行。您也可以選擇執行 Lambda 函數作為不隔離的程序（即無容器模式）。

在 Linux 程序組態下，對於隔離模式，從下列選項中選擇 Lambda 函數的容器化：

- Greengrass 容器 – Lambda 函數會在容器中執行。這是預設選擇。
- 無容器 – Lambda 函數執行為沒有任何隔離的程序。

如果您在容器中執行 Lambda 函數，請完成下列步驟來設定 Lambda 函數的程序組態。

1. 設定記憶體和系統資源的數量，例如磁碟區和裝置，以便可供容器使用。

在容器參數下，執行下列動作。

- a. 針對記憶體大小，輸入您要配置給容器的記憶體大小。您可以指定記憶體大小，以 MB 或 kB 為單位。
 - b. 對於唯讀 `sys` 資料夾，選擇容器是否可以從裝置/`sys` 資料夾讀取資訊。預設值為 `False`。
2. （選用）設定容器化 Lambda 函數可存取的本機磁碟區。當您定義磁碟區時，AWS IoT Greengrass 核心軟體會將來源檔案掛載到容器內的目的地。

- a. 在磁碟區下，選擇新增磁碟區。
 - b. 針對您新增的每個磁碟區，指定下列選項：
 - 實體磁碟區 – 核心裝置上的來源資料夾路徑。
 - 邏輯磁碟區 – 容器中目的地資料夾的路徑。
 - 許可 – (選用) 從容器存取來源資料夾的許可。您可以從以下選項中選擇：
 - 唯讀 – Lambda 函數具有來源資料夾的唯讀存取權。這是預設選擇。
 - 讀寫 – Lambda 函數具有來源資料夾的讀寫存取權。
 - 新增群組擁有者 – (選用) 是否要將執行 Lambda 函數元件的系統群組新增為來源資料夾的擁有者。預設值為 False。
 - c. 若要移除磁碟區，請選擇您要移除的磁碟區旁的移除。
3. (選用) 設定容器化 Lambda 函數可存取的本機系統裝置。
 - a. 在裝置下，選擇新增裝置。
 - b. 針對您新增的每個裝置，指定下列選項：
 - 掛載路徑 – 核心裝置上的系統裝置的路徑。
 - 許可 – (選用) 從容器存取系統裝置的許可。您可以從以下選項中選擇：
 - 唯讀 – Lambda 函數具有系統裝置的唯讀存取權。這是預設選擇。
 - 讀寫 – Lambda 函數具有來源資料夾的讀寫存取權。
 - 新增群組擁有者 – (選用) 是否要將執行 Lambda 函數元件的系統群組新增為系統裝置的擁有者。預設值為 False。

步驟 6：建立 Lambda 函數元件

設定 Lambda 函數元件的設定後，請選擇建立以完成建立新元件。

若要在核心裝置上執行 Lambda 函數，您可以將新元件部署至核心裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

將 Lambda 函數匯入為元件 (AWS CLI)

使用 [CreateComponentVersion](#) 操作從 Lambda 函數建立元件。當您呼叫此操作時，請指定 `lambdaFunction` 以匯入 Lambda 函數。

任務

- [步驟 1：定義 Lambda 函數組態](#)
- [步驟 2：建立 Lambda 函數元件](#)

步驟 1：定義 Lambda 函數組態

1. 建立名為 `lambda-function-component.json` 的檔案，然後將下列 JSON 物件複製到 檔案。將 `lambdaArn` 取代為要匯入之 Lambda 函數的 ARN。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
  }
}
```

Important

您必須指定包含要匯入之函數版本的 ARN。您不能使用 `$LATEST` 之類的版本別名。

2. (選用) 指定元件的名稱 (`componentName`)。如果您省略此參數，會使用 Lambda 函數的名稱 AWS IoT Greengrass 建立元件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda"
  }
}
```

3. (選用) 指定元件的版本 (`componentVersion`)。如果您省略此參數，會使用 Lambda 函數的版本 AWS IoT Greengrass 建立元件，做為有效的語意版本。例如，如果您的函數版本為 3，則元件版本會變成 3.0.0。

Note

您上傳的每個元件版本都必須是唯一的。請務必上傳正確的元件版本，因為您在上傳之後無法編輯。

AWS IoT Greengrass 使用元件的語意版本。語意版本遵循 major.minor.patch 號碼系統。例如，版本1.0.0代表元件的第一個主要版本。如需詳細資訊，請參閱[語意版本規格](#)。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0"
  }
}
```

4. (選用) 指定此 Lambda 函數支援的平台。每個平台都包含識別平台的屬性映射。所有核心裝置都有作業系統 (os) 和架構 (architecture) 的屬性。Core AWS IoT Greengrass 軟體可能會新增其他平台屬性。您也可以將 [Greengrass 核心元件](#) 部署至核心裝置時指定自訂平台屬性。請執行下列操作：

- a. 將平台清單 (componentPlatforms) 新增至 中的 Lambda 函數 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. 將每個支援的平台新增至清單。每個平台都有一個易於 name 識別的 和屬性映射。下列範例指定此函數支援執行 Linux 的 x86 裝置。

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

```
}
```

您的 `lambda-function-component.json` 可能包含類似下列範例的文件。

```
{
  "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. (選用) 指定 Lambda 函數的元件相依性。當您部署 Lambda 函數元件時，部署會包含要讓函數執行的這些相依性。

Important

若要匯入您建立以在 AWS IoT Greengrass V1 上執行的 Lambda 函數，您必須為函數使用的功能定義個別元件相依性，例如秘密、本機陰影和串流管理員。將這些元件定義為[硬相依性](#)，以便在相依性變更狀態時重新啟動 Lambda 函數元件。如需詳細資訊，請參閱[匯入 V1 Lambda 函數](#)。

請執行下列操作：

- a. 將元件相依性 (`componentDependencies`) 的映射新增至 中的 Lambda 函數 `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. 將每個元件相依性新增至映射。指定元件名稱做為金鑰，並使用下列參數指定物件：

- **versionRequirement** – 識別元件相依性相容版本的 npm 樣式語意版本限制。您可以指定單一版本或一系列版本。如需語意版本限制的詳細資訊，請參閱 [npm 轉換器計算器](#)。
- **dependencyType** – (選用) 相依性的類型。請選擇下列項目：
 - **SOFT** – 如果相依性變更狀態，Lambda 函數元件不會重新啟動。
 - **HARD** – 如果相依性變更狀態，Lambda 函數元件會重新啟動。

預設值為 **HARD**。

下列範例指定此 Lambda 函數取決於[串流管理員元件](#)第一個主要版本中的任何版本。當串流管理員重新啟動或更新時，Lambda 函數元件會重新啟動。

```

{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}

```

您的 `lambda-function-component.json` 可能包含類似下列範例的文件。

```

{
  "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. (選用) 設定 Lambda 函數參數以用來執行函數。您可以設定選項，例如環境變數、訊息事件來源、逾時和容器設定。請執行下列操作：
 - a. 將 Lambda 參數物件 (componentLambdaParameters) 新增至 中的 Lambda 函數 `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. (選用) 指定 Lambda 函數訂閱工作訊息的事件來源。您可以指定事件來源，將此函數訂閱本機發佈/訂閱訊息和 AWS IoT Core MQTT 訊息。從事件來源接收訊息時，會呼叫 Lambda 函數。

 Note

若要將此函數訂閱其他 Lambda 函數或元件的訊息，請在部署此 Lambda 函數元件時部署 [舊版訂閱路由器](#) 元件。部署舊版訂閱路由器元件時，請指定 Lambda 函數使用的訂閱。

請執行下列操作：

- i. 將事件來源清單 (eventSources) 新增至 Lambda 函數參數。

```

{
  "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. 將每個事件來源新增至清單。每個事件來源都有下列參數：

- topic – 要訂閱訊息的主題。
- type – 事件來源的類型。您可以從以下選項中選擇：
 - PUB_SUB - 訂閱本機發佈/訂閱訊息。

如果您使用 [Greengrass nucleus](#) v2.6.0 或更新版本，以及 [Lambda Manager](#) v2.2.5 或更新版本，您可以在指定此類型topic時使用 中的 MQTT 主題萬用字元 (+ 和 #)。

- IOT_CORE – 訂閱 AWS IoT Core MQTT 訊息。

當您指定此類型topic時，您可以在 中使用 MQTT 主題萬用字元 (+ 和 #)。

下列範例會針對符合主題篩選條件hello/world/+的主題訂閱 AWS IoT Core MQTT。

```

{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}

```

您的 lambda-function-component.json 可能看起來類似下列範例。

```

{
  "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. (選用) 在 Lambda 函數參數物件中指定下列任何參數：

- `environmentVariables` – Lambda 函數執行時可用的環境變數映射。
- `execArgs` – 執行時要傳遞給 Lambda 函數的引數清單。
- `inputPayloadEncodingType` – Lambda 函數支援的承載類型。您可以從以下選項中選擇：
 - `json`
 - `binary`

預設：`json`

- `pinned` – 是否固定 Lambda 函數。預設值為 `true`。
 - 固定 (或長期) Lambda 函數會在 AWS IoT Greengrass 啟動時啟動，並在自己的容器中持續執行。

- 非固定（或隨需）Lambda 函數只會在接收工作項目時啟動，並在閒置達指定的閒置時間上限後結束。如果函數有多個工作項目，則 AWS IoT Greengrass Core 軟體會建立函數的多個執行個體。

使用 `maxIdleTimeInSeconds` 來設定函數的閒置時間上限。

- `timeoutInSeconds` – Lambda 函數在逾時之前可執行的秒數上限。預設為 3 秒。
- `statusTimeoutInSeconds` – Lambda 函數元件將狀態更新傳送至 Lambda 管理員元件的間隔，以秒為單位。此參數僅適用於固定函數。預設值為 60 秒。
- `maxIdleTimeInSeconds` – 在 AWS IoT Greengrass 核心軟體停止其程序之前，非固定 Lambda 函數可以閒置的時間上限，以秒為單位。預設值為 60 秒。
- `maxInstancesCount` – 非固定 Lambda 函數可同時執行的執行個體數目上限。預設值為 100 個執行個體。
- `maxQueueSize` – Lambda 函數元件的訊息佇列大小上限。Core AWS IoT Greengrass 軟體會將訊息存放在 FIFO first-in-first-out 佇列中，直到可以執行 Lambda 函數來取用每個訊息為止。預設值為 1,000 則訊息。

您的 `lambda-function-component.json` 可能包含類似下列範例的文件。

```
{
  "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. (選用) 設定 Lambda 函數的容器設定。根據預設，Lambda 函數會在 AWS IoT Greengrass Core 軟體內的隔離執行期環境中執行。您也可以選擇執行 Lambda 函數作為程序，無需任何隔離。如果您在容器中執行 Lambda 函數，您可以設定容器的記憶體大小，以及 Lambda 函數可用的系統資源。請執行下列操作：
 - i. 將 Linux 程序參數物件 (linuxProcessParams) 新增至 中的 Lambda 參數物件 `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. (選用) 指定 Lambda 函數是否在容器中執行。將 `isolationMode` 參數新增至程序參數物件，然後從下列選項中選擇：

- GreengrassContainer – Lambda 函數會在容器中執行。
- NoContainer – Lambda 函數會以無任何隔離的程序執行。

預設值為 GreengrassContainer。

- iii. (選用) 如果您在容器中執行 Lambda 函數，您可以設定記憶體數量和系統資源，例如磁碟區和裝置，以便可供容器使用。請執行下列操作：

- A. 將容器參數物件 (containerParams) 新增至 中的 Linux 程序參數物件 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. (選用) 新增 `memorySizeInKB` 參數以指定容器的記憶體大小。預設值為 16,384 KB (16 MB)。
- C. (選用) 新增 `mountR0Sysfs` 參數，以指定容器是否可以從裝置/`sys`資料夾讀取資訊。預設值為 `false`。
- D. (選用) 設定容器化 Lambda 函數可存取的本機磁碟區。當您定義磁碟區時，AWS IoT Greengrass 核心軟體會將來源檔案掛載到容器內的目的地。請執行下列操作：
 - I. 將磁碟區清單 (volumes) 新增至容器參數。

```

{
  "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. 將每個磁碟區新增至清單。每個磁碟區都有下列參數：

- `sourcePath` – 核心裝置上的來源資料夾路徑。
- `destinationPath` – 容器中目的地資料夾的路徑。
- `permission` – (選用) 從容器存取來源資料夾的許可。您可以從以下選項中選擇：
 - `ro` – Lambda 函數具有來源資料夾的唯讀存取權。
 - `rw` – Lambda 函數具有來源資料夾的讀寫存取權。

預設值為 `ro`。

- `addGroupOwner` – (選用) 是否要將執行 Lambda 函數元件的系統群組新增為來源資料夾的擁有者。預設值為 `false`。

您的 `lambda-function-component.json` 可能包含類似下列範例的文件。

```
{
  "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. (選用) 設定容器化 Lambda 函數可存取的本機系統裝置。請執行下列操作：

I. 將系統裝置清單 (devices) 新增至容器參數。

```

{
  "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. 將每個系統裝置新增至清單。每個系統裝置都有下列參數：

- `path` – 核心裝置上的系統裝置的路徑。
- `permission` – (選用) 從容器存取系統裝置的許可。您可以從以下選項中選擇：
 - `ro` – Lambda 函數具有系統裝置的唯讀存取權。
 - `rw` – Lambda 函數具有系統裝置的讀寫存取權。

預設值為 `ro`。

- `addGroupOwner` – (選用) 是否要將執行 Lambda 函數元件的系統群組新增為系統裝置的擁有者。預設值為 `false`。

您的 `lambda-function-component.json` 可能包含類似下列範例的文件。

```
{
  "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. (選用) 為元件新增標籤 (tags)。如需詳細資訊，請參閱[標記您的 AWS IoT Greengrass Version 2 資源](#)。

步驟 2：建立 Lambda 函數元件

1. 執行下列命令，從 建立 Lambda 函數元件 `lambda-function-component.json`。

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
  "status": {
    "componentState": "REQUESTED",
    "message": "NONE",
    "errors": {}
  }
}
```

arn 從輸出複製，以在下一個步驟中檢查元件的狀態。

2. 當您建立元件時，其狀態為 REQUESTED。然後，AWS IoT Greengrass 驗證元件可部署。您可以執行下列命令來查詢元件狀態，並確認元件可部署。將 取代arn為上一個步驟的 ARN。

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

如果元件驗證，回應會指出元件狀態為 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"
      }
    }
  ]
}
```

元件為 之後DEPLOYABLE，您可以將 Lambda 函數部署至核心裝置。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和通訊 AWS IoT Core

核心裝置上執行的元件可以使用中的 AWS IoT Greengrass 核心程序間通訊 (IPC) 程式庫 AWS IoT Device SDK，與核和其他 Greengrass AWS IoT Greengrass 元件進行通訊。若要開發和執行使用 IPC 的自訂元件，您必須使用 AWS IoT Device SDK 連線到 AWS IoT Greengrass Core IPC 服務並執行 IPC 操作。

IPC 介面支援兩種類型的操作：

- 請求/回應

元件會將請求傳送至 IPC 服務，並接收包含請求結果的回應。

- 訂閱

元件會將訂閱請求傳送至 IPC 服務，並預期事件訊息串流回應。元件提供訂閱處理常式，可處理事件訊息、錯誤和串流關閉。AWS IoT Device SDK 包含處理常式界面，具有每個 IPC 操作的正確回應和事件類型。如需詳細資訊，請參閱[訂閱 IPC 事件串流](#)。

主題

- [IPC 用戶端版本](#)
- [支援用於程序間通訊 SDKs](#)
- [連線至 AWS IoT Greengrass Core IPC 服務](#)
- [授權元件執行 IPC 操作](#)
- [訂閱 IPC 事件串流](#)
- [IPC 最佳實務](#)
- [發佈/訂閱本機訊息](#)
- [發佈/訂閱 AWS IoT Core MQTT 訊息](#)
- [與元件生命週期互動](#)
- [與元件組態互動](#)
- [擷取秘密值](#)
- [與本機陰影互動](#)
- [管理本機部署和元件](#)

- [驗證和授權用戶端裝置](#)

IPC 用戶端版本

在 Java 和 Python SDKs 的更新版本中，AWS IoT Greengrass 提供了 IPC 用戶端的改良版本，稱為 IPC 用戶端 V2。IPC 用戶端 V2：

- 減少您需要寫入以使用 IPC 操作的程式碼量，並協助避免 IPC 用戶端 V1 可能發生的常見錯誤。
- 在單獨的執行緒中呼叫訂閱處理常式回呼，因此您現在可以在訂閱處理常式回呼中執行封鎖程式碼，包括額外的 IPC 函數呼叫。IPC 用戶端 V1 使用相同的執行緒與 IPC 伺服器通訊，並呼叫訂閱處理常式回呼。
- 可讓您使用 Lambda 表達式 (Java) 或函數 (Python) 呼叫訂閱操作。IPC 用戶端 V1 需要您定義訂閱處理常式類別。
- 提供每個 IPC 操作的同步和非同步版本。IPC 用戶端 V1 僅提供每個操作的非同步版本。

我們建議您使用 IPC 用戶端 V2 來利用這些改進。不過，本文件和某些線上內容中的許多範例只會示範如何使用 IPC 用戶端 V1。您可以使用下列範例和教學課程來查看使用 IPC 用戶端 V2 的範例元件：

- [PublishToTopic 範例](#)
- [SubscribeToTopic 範例](#)
- [教學課程：開發可延遲元件更新的 Greengrass 元件](#)
- [教學課程：透過 MQTT 與本機 IoT 裝置互動](#)

目前，AWS IoT Device SDK for C++ v2 僅支援 IPC 用戶端 V1。

支援用於程序間通訊 SDKs

AWS IoT Greengrass 核心 IPC 程式庫包含在下列 AWS IoT Device SDK 版本中。

SDK	最低版本	Usage	
AWS IoT Device SDK for Java v2	v1.6.0	請參閱 AWS IoT Device SDK 用於 Java v2 (IPC 用戶端 V2)	

SDK	最低版本	Usage
AWS IoT Device SDK for Python v2	1.9.0 版	請參閱 AWS IoT Device SDK 適用於 Python v2 (IPC 用戶端 V2)
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	請參閱 AWS IoT Device SDK 用於 C++ v2
AWS IoT Device SDK for JavaScript v2	v1.12.0	請參閱 AWS IoT Device SDK 適用於 JavaScript v2 (IPC 用戶端 V1)

連線至 AWS IoT Greengrass Core IPC 服務

若要在自訂元件中使用程序間通訊，您必須建立與 AWS IoT Greengrass 核心軟體執行之 IPC 伺服器通訊端的連線。完成下列任務，以您選擇的 AWS IoT Device SDK 語言下載並使用。

AWS IoT Device SDK 用於 Java v2 (IPC 用戶端 V2)

使用 AWS IoT Device SDK for Java v2 (IPC 用戶端 V2)

1. 下載 [AWS IoT Device SDK for Java v2](#) (v1.6.0 或更新版本)。
2. 執行下列其中一項以在元件中執行自訂程式碼：
 - 將元件建置為包含的 JAR 檔案 AWS IoT Device SDK，然後在元件配方中執行此 JAR 檔案。
 - 將 AWS IoT Device SDK JAR 定義為元件成品，並在元件配方中執行應用程式時將該成品新增至 classpath。
3. 使用下列程式碼來建立 IPC 用戶端。

```
try (GreengrassCoreIPCClientV2 ipcClient =
    GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
}
```

```
    System.exit(1);  
}
```

AWS IoT Device SDK 適用於 Python v2 (IPC 用戶端 V2)

使用 AWS IoT Device SDK for Python v2 (IPC 用戶端 V2)

1. 下載 [AWS IoT Device SDK for Python](#) (v1.9.0 或更新版本)。
2. 將開發套件的[安裝步驟](#)新增至元件配方中的安裝生命週期。
3. 建立與 AWS IoT Greengrass Core IPC 服務的連線。使用下列程式碼來建立 IPC 用戶端。

```
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)
```

AWS IoT Device SDK 用於 C++ v2

若要建置適用於 C++ 的 AWS IoT Device SDK v2，裝置必須具有下列工具：

- C++ 11 或更新版本
- CMake 3.1 或更新版本
- 下列其中一個編譯器：
 - GCC 4.8 或更新版本
 - Clang 3.9 或更新版本
 - MSVC 2015 或更新版本

使用 AWS IoT Device SDK 適用於 C++ v2 的

1. 下載 [AWS IoT Device SDK for C++ v2](#) (v1.17.0 或更新版本)。
2. 遵循 [README 中的安裝說明](#)，從來源建置 AWS IoT Device SDK 適用於 C++ v2 的。

3. 在 C++ 建置工具中，連結您在上一個步驟中建置的 Greengrass IPC 程式庫 `AWS::GreengrassIpc-cpp`。下列 `CMakeLists.txt` 範例會將 Greengrass IPC 程式庫連結至您使用 CMake 建置的專案。

```
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. 在元件程式碼中，建立與 AWS IoT Greengrass Core IPC 服務的連線，以建立 IPC 用戶端 (`Aws::Greengrass::GreengrassCoreIpcClient`)。您必須定義 IPC 連線生命週期處理常式，以處理 IPC 連線、中斷連線和錯誤事件。下列範例會建立 IPC 用戶端和 IPC 連線生命週期處理常式，在 IPC 用戶端連線、中斷連線和遇到錯誤時列印。

```
#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. 若要在元件中執行自訂程式碼，請將程式碼建置為二進位成品，然後在元件配方中執行二進位成品。將成品的Execute許可設定為 OWNER，讓 AWS IoT Greengrass Core 軟體執行二進位成品。

元件配方的Manifests區段可能看起來類似下列範例。

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
```

AWS IoT Device SDK 適用於 JavaScript v2 (IPC 用戶端 V1)

若要建置 AWS IoT Device SDK 適用於 JavaScript v2 的以搭配 NodeJS 使用，裝置必須具備下列工具：

- NodeJS 10.0 或更新版本
 - 執行 `node -v` 以檢查節點版本。
- CMake 3.1 或更新版本

使用 AWS IoT Device SDK for JavaScript v2 (IPC 用戶端 V1)

1. 下載 [AWS IoT Device SDK for JavaScript v2](#) (v1.12.10 或更新版本)。
2. 遵循 [README 中的安裝說明](#)，從來源建置 AWS IoT Device SDK 適用於 JavaScript v2 的。
3. 建立與 AWS IoT Greengrass Core IPC 服務的連線。完成下列步驟以建立 IPC 用戶端並建立連線。
4. 使用下列程式碼來建立 IPC 用戶端。

```
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2';

let client = greengrasscoreipc.createClient();
```

5. 使用下列程式碼，從您的元件建立與 Greengrass 核的連線。

```
await client.connect();
```

授權元件執行 IPC 操作

若要允許自訂元件使用某些 IPC 操作，您必須定義授權政策，允許元件對特定資源執行操作。每個授權政策都會定義操作清單，以及政策允許的資源清單。例如，發佈/訂閱訊息 IPC 服務會定義主題資源的發佈和訂閱操作。您可以使用 * 萬用字元來允許存取所有操作或所有資源。

您可以使用 `accessControl` 組態參數來定義授權政策，您可以在元件配方中或部署元件時加以設定。`accessControl` 物件會將 IPC 服務識別符映射至授權政策清單。您可以為每個 IPC 服務定義多個授權政策來控制存取。每個授權政策都有一個政策 ID，在所有元件中必須是唯一的。

Tip

若要建立唯一的政策 IDs，您可以結合元件名稱、IPC 服務名稱和計數器。例如，名為 `com.example.HelloWorld` 的元件可能會使用下列 IDs 定義兩個發佈/訂閱授權政策：

- `com.example.HelloWorld:pubsub:1`
- `com.example.HelloWorld:pubsub:2`

授權政策使用以下格式。此物件是 `accessControl` 組態參數。

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
```

授權政策中的萬用字元

您可以在 IPC 授權政策的 `resources` 元素中使用 * 萬用字元，以允許存取單一授權政策中的多個資源。

- 在所有版本的 [Greengrass 核](#) 中，您可以指定單一 * 字元做為資源，以允許存取所有資源。
- 在 [Greengrass 核](#) v2.6.0 和更新版本中，您可以在資源中指定 * 字元，以符合字元的任意組合。例如，您可以指定 `factory/1/devices/Thermostat*/status` 允許存取工廠中所有恆溫器裝置的狀態主題，其中每個裝置的名稱開頭為 `Thermostat`。

當您定義 AWS IoT Core MQTT IPC 服務的授權政策時，您也可以使用 MQTT 萬用字元 (+ 和 #) 來比對多個資源。如需詳細資訊，請參閱 [MQTT IPC 授權政策中的 AWS IoT Core MQTT 萬用字元](#)。

授權政策中的配方變數

如果您使用 [Greengrass 核](#) v2.6.0 或更新版本，並將 Greengrass 核的 [interpolateComponentConfiguration](#) 組態選項設定為 `true`，您可以在授權政策中使用 `{iot:thingName}` [配方變數](#)。當您需要包含核心裝置名稱的授權政策時，例如 MQTT 主題或裝置影子，您可以使用此配方變數來設定一組核心裝置的單一授權政策。例如，您可以允許元件存取陰影 IPC 操作的下列資源。

```
$aws/things/{iot:thingName}/shadow/
```

授權政策中的特殊字元

若要在授權政策中指定文字*或?字元，您必須使用逸出序列。下列逸出序列會指示 AWS IoT Greengrass Core 軟體使用常值，而非字元的特殊意義。例如，* 字元是符合任何字元組合的[萬用字元](#)。

文字字元	逸出序列	備註
*	<code>\${*}</code>	
?	<code>\${?}</code>	AWS IoT Greengrass 目前不支援萬?用字元，符合任何單一字元。
\$	<code>\${\$}</code>	使用此逸出序列來比對包含的資源\${。例如，若要符合名為的資源\${resourceName}，您必須指定 <code>\${\${resourceName}}</code> 。否則，若要比對包含的資源\$，您可以使用常值\$，例如 允許存取開頭為的主題\$aws。

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example具有 授權政策的元件配方範例

下列範例元件配方包含 `accessControl` 物件，可定義授權政策。此政策授權 `com.example>HelloWorld`元件發佈至 `test/topic`主題。

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使用授權政策更新元件組態的範例

部署中的下列範例組態更新指定 使用定義授權政策的 `accessControl` 物件來設定元件。此政策授權 `com.example.HelloWorld` 元件發佈至 `test/topic` 主題。

Console

要合併的組態

```

{
  "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

下列命令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-deployment.json
```

`hello-world-deployment.json` 檔案包含下列 JSON 文件。

```
{
  "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

下列 [Greengrass CLI](#) 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

hello-world-configuration.json 檔案包含下列 JSON 文件。

```
{
  "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"
            ]
          }
        }
      }
    }
  }
}
```

```
}  
}  
}  
}  
}
```

訂閱 IPC 事件串流

您可以使用 IPC 操作來訂閱 Greengrass 核心裝置上的事件串流。若要使用訂閱操作，請定義訂閱處理常式並建立 IPC 服務的請求。然後，每次核心裝置將事件訊息串流到您的元件時，IPC 用戶端都會執行訂閱處理常式的函數。

您可以關閉訂閱以停止處理事件訊息。若要這麼做，請在您用來開啟訂閱的訂閱操作物件上呼叫 `closeStream()`(Java)、`close()` (Python) 或 `Close()`(C++)。

AWS IoT Greengrass Core IPC 服務支援下列訂閱操作：

- [SubscribeToTopic](#)
- [SubscribeToIoTCore](#)
- [SubscribeToComponentUpdates](#)
- [SubscribeToConfigurationUpdate](#)
- [SubscribeToValidateConfigurationUpdates](#)

主題

- [定義訂閱處理常式](#)
- [範例訂閱處理常式](#)

定義訂閱處理常式

若要定義訂閱處理常式，請定義處理事件訊息、錯誤和串流關閉的回呼函數。如果您使用 IPC 用戶端 V1，則必須在類別中定義這些函數。如果您使用適用於 Java 和 Python SDKs 較新版本的 IPC 用戶端 V2，您可以定義這些函數，而無需建立訂閱處理常式類別。

Java

如果您使用 IPC 用戶端 V1，則必須實作一

般 `software.amazon.awssdk.eventstreamrpc.StreamResponseHandler` <*StreamEventType*>

面。*StreamEventType* 是訂閱操作的事件訊息類型。定義下列函數來處理事件訊息、錯誤和串流關閉。

如果您使用 IPC 用戶端 V2，您可以在訂閱處理常式類別之外定義這些函數，或使用 [lambda 運算式](#)。

```
void onStreamEvent(StreamEventType event)
```

IPC 用戶端在收到事件訊息時呼叫的回呼，例如 MQTT 訊息或元件更新通知。

```
boolean onStreamError(Throwable error)
```

IPC 用戶端在發生串流錯誤時呼叫的回呼。

傳回 true 以關閉訂閱串流，因為發生錯誤，或傳回 false 以保持串流開啟。

```
void onStreamClosed()
```

IPC 用戶端在串流關閉時呼叫的回呼。

Python

如果您使用 IPC 用戶端 V1，則必須擴展對應至訂閱操作的串流回應處理常式類別。AWS IoT Device SDK 包含每個訂閱操作的訂閱處理常式類別。*StreamEventType* 是訂閱操作的事件訊息類型。定義下列函數來處理事件訊息、錯誤和串流關閉。

如果您使用 IPC 用戶端 V2，您可以在訂閱處理常式類別之外定義這些函數，或使用 [lambda 運算式](#)。

```
def on_stream_event(self, event: StreamEventType) -> None
```

IPC 用戶端在收到事件訊息時呼叫的回呼，例如 MQTT 訊息或元件更新通知。

```
def on_stream_error(self, error: Exception) -> bool
```

IPC 用戶端在發生串流錯誤時呼叫的回呼。

傳回 true 以關閉訂閱串流，因為發生錯誤，或傳回 false 以保持串流開啟。

```
def on_stream_closed(self) -> None
```

IPC 用戶端在串流關閉時呼叫的回呼。

C++ (IPC client V1)

實作衍生自對應至訂閱操作之串流回應處理常式類別的類別。AWS IoT Device SDK 包含每個訂閱操作的訂閱處理常式基本類別。*StreamEventType* 是訂閱操作的事件訊息類型。定義下列函數來處理事件訊息、錯誤和串流關閉。

```
void OnStreamEvent(StreamEventType *event)
```

IPC 用戶端在收到事件訊息時呼叫的回呼，例如 MQTT 訊息或元件更新通知。

```
bool OnStreamError(OnError *error)
```

IPC 用戶端在發生串流錯誤時呼叫的回呼。

傳回 true 以關閉訂閱串流，因為發生錯誤，或傳回 false 以保持串流開啟。

```
void OnStreamClosed()
```

IPC 用戶端在串流關閉時呼叫的回呼。

JavaScript

實作衍生自對應至訂閱操作之串流回應處理常式類別的類別。AWS IoT Device SDK 包含每個訂閱操作的訂閱處理常式基本類別。*StreamEventType* 是訂閱操作的事件訊息類型。定義下列函數來處理事件訊息、錯誤和串流關閉。

```
on(event: 'ended', listener: StreamingOperationEndedListener)
```

IPC 用戶端在串流關閉時呼叫的回呼。

```
on(event: 'streamError', listener: StreamingRpcErrorListener)
```

IPC 用戶端在發生串流錯誤時呼叫的回呼。

傳回 true 以關閉訂閱串流，因為發生錯誤，或傳回 false 以保持串流開啟。

```
on(event: 'message', listener: (message: InboundMessageType) => void)
```

IPC 用戶端在收到事件訊息時呼叫的回呼，例如 MQTT 訊息或元件更新通知。

範例訂閱處理常式

下列範例示範如何使用 [SubscribeToTopic](#) 操作和訂閱處理常式來訂閱本機發佈/訂閱訊息。

Java (IPC client V2)

Example範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```

#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範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```
#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範例：訂閱本機發佈/訂閱訊息

```

#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);
    }
}
```

IPC 最佳實務

在自訂元件中使用 IPC 的最佳實務在 IPC 用戶端 V1 和 IPC 用戶端 V2 之間有所不同。遵循您使用的 IPC 用戶端版本的最佳實務。

IPC client V2

IPC 用戶端 V2 會在單獨的執行緒中執行回呼函數，因此相較於 IPC 用戶端 V1，當您使用 IPC 和寫入訂閱處理常式函數時，必須遵循的準則較少。

- 重複使用一個 IPC 用戶端

建立 IPC 用戶端之後，請保持開啟狀態，並重複使用於所有 IPC 操作。建立多個用戶端會使用額外的資源，並可能導致資源洩漏。

- 處理例外狀況

IPC 用戶端 V2 會記錄訂閱處理常式函數中未攔截的例外狀況。您應該在處理常式函數中擷取例外狀況，以處理程式碼中發生的錯誤。

IPC client V1

IPC 用戶端 V1 使用與 IPC 伺服器通訊的單一執行緒，並呼叫訂閱處理常式。當您寫入訂閱處理常式函數時，必須考慮此同步行為。

- 重複使用一個 IPC 用戶端

建立 IPC 用戶端之後，請保持開啟狀態，並重複使用於所有 IPC 操作。建立多個用戶端會使用額外的資源，並可能導致資源洩漏。

- 以非同步方式執行封鎖碼

IPC 用戶端 V1 無法在執行緒遭到封鎖時傳送新請求或處理新事件訊息。您應該在從處理常式函數執行的個別執行緒中執行封鎖碼。封鎖程式碼包含sleep呼叫、持續執行的迴圈，以及需要時間才能完成的同步 I/O 請求。

- 以非同步方式傳送新的 IPC 請求

IPC 用戶端 V1 無法從訂閱處理常式函數內傳送新請求，因為如果您等待回應，請求會封鎖處理常式函數。您應該在從處理常式函數執行的個別執行緒中傳送 IPC 請求。

- 處理例外狀況

IPC 用戶端 V1 不會處理訂閱處理常式函數中未攔截的例外狀況。如果您的處理常式函數擲回例外狀況，訂閱會關閉，而且例外狀況不會出現在您的元件日誌中。您應該在處理常式函數中擷取例外狀況，以保持訂閱開啟，並記錄程式碼中發生的錯誤。

發佈/訂閱本機訊息

發佈/訂閱 (pubsub) 訊息可讓您傳送和接收主題的訊息。元件可以將訊息發佈至主題，以傳送訊息至其他元件。然後，訂閱該主題的元件可以對其接收的訊息採取行動。

Note

您無法使用此發佈/訂閱 IPC 服務來發佈或訂閱 AWS IoT Core MQTT。如需如何與 AWS IoT Core MQTT 交換訊息的詳細資訊，請參閱 [發佈/訂閱 AWS IoT Core MQTT 訊息](#)。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [PublishToTopic](#)
- [SubscribeToTopic](#)
- [範例](#)

最低 SDK 版本

下表列出在本機主題之間發佈和訂閱訊息時 AWS IoT Device SDK，您必須使用的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	v1.2.10	
AWS IoT Device SDK for Python v2	1.5.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

Authorization

若要在自訂元件中使用本機發佈/訂閱訊息，您必須定義授權政策，允許您的元件向主題傳送和接收訊息。如需定義授權政策的資訊，請參閱 [授權元件執行 IPC 操作](#)。

發佈/訂閱訊息的授權政策具有下列屬性。

IPC 服務識別符：`aws.greengrass.ipc.pubsub`

作業	Description	Resources
<code>aws.greengrass#PublishToTopic</code>	允許元件發佈訊息到您指定的主題。	主題字串，例如 <code>test/topic</code>。使用 <code>*</code> 來比對主題中任意字元組合。 此主題字串不支援 MQTT 主題萬用字元 (<code>#</code> 和 <code>+</code>)。
<code>aws.greengrass#SubscribeToTopic</code>	允許元件訂閱您指定主題的訊息。	主題字串，例如 <code>test/topic</code>。使用 <code>*</code> 來比對主題中任意字元組合。 在 Greengrass nucleus v2.6.0 和更新版本中，您可以訂閱包含 MQTT 主題萬用字元 (<code>#</code> 和 <code>+</code>) 的主題。此主題字串支援 MQTT 主題萬用字元做為文字字元。例如，如果元件的授權政策授予的存取權 <code>test/topic/#</code>，則該元件可以訂閱 <code>test/topic/#</code>，但無法訂閱 <code>test/topic/filter</code>。
<code>*</code>	允許元件發佈和訂閱您指定主題的訊息。	主題字串，例如 <code>test/topic</code>。使用 <code>*</code> 來比對主題中任意字元組合。 在 Greengrass nucleus v2.6.0 和更新版本中，您可以訂閱包

作業	Description	Resources
		含 MQTT 主題萬用字元 (# 和 +) 的主題。此主題字串支援 MQTT 主題萬用字元做為文字字元。例如，如果元件的授權政策授予的存取權 <code>test/topic/c/#</code> ，則該元件可以訂閱 <code>test/topic/#</code> ，但無法訂閱 <code>test/topic/filter</code> 。

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example 範例授權政策

下列範例授權政策允許元件發佈和訂閱所有主題。

```
{
  "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

發布訊息至主題。

請求

此操作的請求具有下列參數：

`topic`

要發佈訊息的主題。

`publishMessage` (Python : `publish_message`)

要發佈的訊息。此物件 `PublishMessage` 包含下列資訊。您必須指定其中一個 `jsonMessage` 和 `binaryMessage`。

`jsonMessage` (Python : `json_message`)

(選用) JSON 訊息。此物件 `JsonMessage` 包含下列資訊：

`message`

JSON 訊息做為 物件。

`context`

訊息的內容，例如發佈訊息的主題。

此功能適用於 [Greengrass 核元件](#) 的 v2.6.0 和更新版本。下表列出 AWS IoT Device SDK 您必須用來存取訊息內容的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.9.3 版	
AWS IoT Device SDK for Python v2	1.11.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.18.4 版	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

 Note

AWS IoT Greengrass 核心軟體在 PublishToTopic 和 SubscribeToTopic 操作中使用相同的訊息物件。AWS IoT Greengrass 核心軟體會在您訂閱時於訊息中設定此內容物件，並在您發佈的訊息中忽略此內容物件。

此物件 MessageContext 包含下列資訊：

topic

發佈訊息的主題。

binaryMessage (Python : binary_message)

(選用) 二進位訊息。此物件 BinaryMessage 包含下列資訊：

message

Blob 形式的二進位訊息。

context

訊息的內容，例如發佈訊息的主題。

此功能適用於 [Greengrass 核元件](#) 的 v2.6.0 和更新版本。下表列出 AWS IoT Device SDK 您必須用來存取訊息內容的最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.9.3 版	
AWS IoT Device SDK for Python v2	1.11.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.18.4 版	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

 Note

AWS IoT Greengrass 核心軟體在 PublishToTopic 和 SubscribeToTopic 操作中使用相同的訊息物件。AWS IoT Greengrass 核心軟體會在您訂閱時於訊息中設定此內容物件，並在您發佈的訊息中忽略此內容物件。

此物件 MessageContext 包含下列資訊：

topic

發佈訊息的主題。

回應

此操作不會在其回應中提供任何資訊。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V2)

Example 範例：發佈二進位訊息

```
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範例：發佈二進位訊息

```

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範例：發佈二進位訊息

```

#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範例：發佈二進位訊息

```

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 getIpClient(){
  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範例：發佈二進位訊息

```
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範例：發佈二進位訊息

```
#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範例：發佈二進位訊息

```
#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

訂閱主題上的訊息。

此操作是您訂閱事件訊息串流的訂閱操作。若要使用此操作，請使用處理事件訊息、錯誤和串流關閉的函數來定義串流回應處理常式。如需詳細資訊，請參閱[訂閱 IPC 事件串流](#)。

事件訊息類型：SubscriptionResponseMessage

請求

此操作的請求具有下列參數：

topic

要訂閱的主題。

Note

在 [Greengrass 核](#) v2.6.0 和更新版本中，此主題支援 MQTT 主題萬用字元 (# 和 +)。

receiveMode (Python : receive_mode)

(選用) 指定元件是否從本身接收訊息的行為。您可以變更此行為，以允許元件對自己的訊息採取動作。預設行為取決於主題是否包含 MQTT 萬用字元。您可以從以下選項中選擇：

- RECEIVE_ALL_MESSAGES – 接收符合主題的所有訊息，包括來自訂閱元件的訊息。

當您訂閱不包含 MQTT 萬用字元的主題時，此模式是預設選項。

- RECEIVE_MESSAGES_FROM_OTHERS – 接收符合主題的所有訊息，但來自訂閱元件的訊息除外。

當您訂閱包含 MQTT 萬用字元的主題時，此模式是預設選項。

此功能適用於 [Greengrass 核元件](#) 的 v2.6.0 和更新版本。下表列出 AWS IoT Device SDK 您必須用來設定接收模式的最低版本。

SDK	最低版本
AWS IoT Device SDK for Java v2	1.9.3 版

SDK	最低版本	
AWS IoT Device SDK for Python v2	1.11.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.18.4 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

回應

此操作的回應包含下列資訊：

messages

訊息串流。此物件 SubscriptionResponseMessage 包含下列資訊。每個訊息都包含 jsonMessage 或 binaryMessage。

jsonMessage (Python：json_message)

(選用) JSON 訊息。此物件 JsonMessage 包含下列資訊：

message

JSON 訊息做為物件。

context

訊息的內容，例如發佈訊息的主題。

此功能適用於 [Greengrass 核元件](#) 的 v2.6.0 和更新版本。下表列出 AWS IoT Device SDK 您必須用來存取訊息內容的最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.9.3 版	
AWS IoT Device SDK for Python v2	1.11.3 版	

SDK	最低版本	
AWS IoT Device SDK 適用於 C++ v2	1.18.4 版	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

 Note

AWS IoT Greengrass 核心軟體在 PublishToTopic 和 SubscribeToTopic 操作中使用相同的訊息物件。AWS IoT Greengrass 核心軟體會在您訂閱時於訊息中設定此內容物件，並在您發佈的訊息中忽略此內容物件。

此物件 MessageContext 包含下列資訊：

topic

發佈訊息的主題。

binaryMessage (Python : binary_message)

(選用) 二進位訊息。此物件 BinaryMessage 包含下列資訊：

message

Blob 形式的二進位訊息。

context

訊息的內容，例如發佈訊息的主題。

此功能適用於 [Greengrass 核心元件](#) 的 v2.6.0 和更新版本。下表列出 AWS IoT Device SDK 您必須用來存取訊息內容的最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.9.3 版	

SDK	最低版本	
AWS IoT Device SDK for Python v2	1.11.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.18.4 版	
適用於 JavaScript 的 AWS IoT Device SDK v2	v1.12.0	

 Note

AWS IoT Greengrass 核心軟體在 PublishToTopic 和 SubscribeToTopic 操作中使用相同的訊息物件。AWS IoT Greengrass 核心軟體會在您訂閱時於訊息中設定此內容物件，並在您發佈的訊息中忽略此內容物件。

此物件 MessageContext 包含下列資訊：

topic

發佈訊息的主題。

topicName (Python : topic_name)

發佈訊息的主題。

 Note

此屬性目前未使用。在 [Greengrass 核](#) v2.6.0 和更新版本中，您可以從 取得 (jsonMessage | binaryMessage).context.topic 值 SubscriptionResponseMessage，以取得訊息發佈的主題。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V2)

Example範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```

#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範例：訂閱本機發佈/訂閱訊息

```
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 getIpClient(){
    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範例：訂閱本機發佈/訂閱訊息

```
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範例：訂閱本機發佈/訂閱訊息

```
#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範例：訂閱本機發佈/訂閱訊息

```

#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);
    }
}

```

範例

使用下列範例來了解如何在元件中使用發佈/訂閱 IPC 服務。

發佈/訂閱發佈者範例 (Java、IPC 用戶端 V1)

下列範例配方允許 元件發佈至所有主題。

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

```

下列範例 Java 應用程式示範如何使用 發佈/訂閱 IPC 服務將訊息發佈至其他元件。

```

/* 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);
    }
}
}

```

發佈/訂閱訂閱者範例 (Java、IPC 用戶端 V1)

下列範例配方允許 元件訂閱所有主題。

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

```

下列範例 Java 應用程式示範如何使用 發佈/訂閱 IPC 服務來訂閱其他元件的訊息。

```

/* 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.");
        }
    }
}

```

發佈/訂閱發佈者範例 (Python、IPC 用戶端 V1)

下列範例配方允許 元件發佈至所有主題。

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

```

下列範例 Python 應用程式示範如何使用 發佈/訂閱 IPC 服務將訊息發佈至其他元件。

```

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)
```

發佈/訂閱訂閱者範例 (Python、IPC 用戶端 V1)

下列範例配方允許 元件訂閱所有主題。

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
```

下列範例 Python 應用程式示範如何使用 發佈/訂閱 IPC 服務來訂閱其他元件的訊息。

```
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)

```

發佈/訂閱發佈者範例 (C++、IPC 用戶端 V1)

下列範例配方允許 元件發佈至所有主題。

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
```

下列 C++ 應用程式範例示範如何使用 發佈/訂閱 IPC 服務將訊息發佈至其他元件。

```
#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;
}

```

發佈/訂閱訂閱者範例 (C++、IPC 用戶端 V1)

下列範例配方允許 元件訂閱所有主題。

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

```

下列 C++ 應用程式範例示範如何使用 發佈/訂閱 IPC 服務來訂閱其他元件的訊息。

```
#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;
}

```

發佈/訂閱發佈者範例 (Rust)

下列範例配方允許 元件發佈至所有主題。

```

{
  "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"
          }
        }
      ]
    }
  ]
}

```

下列範例 Rust 應用程式示範如何使用 發佈/訂閱 IPC 服務將訊息發佈至其他元件。

```

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}");
}

```

發佈/訂閱訂閱者範例 (Rust)

下列範例配方允許 元件訂閱所有主題。

```

{
  "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"
        }
    }
]
}
]
}

```

下列範例 Rust 應用程式示範如何使用 發佈/訂閱 IPC 服務來訂閱來自其他元件的訊息。

```

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));
    }
}

```


}

發佈/訂閱發佈者範例 (C)

下列範例配方允許 元件發佈至所有主題。

```
{
  "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"
          }
        }
      ]
    }
  ]
}
```

```
]
}
```

下列範例 C 應用程式示範如何使用 發佈/訂閱 IPC 服務將訊息發佈至其他元件。

```
#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
    );
}
```

發佈/訂閱訂閱者範例 (C)

下列範例配方允許 元件訂閱所有主題。

```

{
  "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"
          }
        }
      ]
    }
  ]
}

```

下列範例 C 應用程式示範如何使用 發佈/訂閱 IPC 服務來訂閱來自其他元件的訊息。

```
#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);
}

```

發佈/訂閱發佈者範例 (C++、元件 SDK)

下列範例配方允許 元件發佈至所有主題。

```

{
  "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"
                    }
                }
            ]
        }
    ]
}

```

下列 C++ 應用程式範例示範如何使用 發佈/訂閱 IPC 服務將訊息發佈至其他元件。

```

#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";
}

```

發佈/訂閱訂閱者範例 (C++、元件 SDK)

下列範例配方允許 元件訂閱所有主題。

```

{
  "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"
      }
    }
  ]
}

```

下列 C++ 應用程式範例示範如何使用 發佈/訂閱 IPC 服務來訂閱來自其他元件的訊息。

```

#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);
}
}
```

發佈/訂閱 AWS IoT Core MQTT 訊息

AWS IoT Core MQTT 訊息 IPC 服務可讓您在 之間傳送和接收 MQTT 訊息 AWS IoT Core。元件可以將訊息發佈到 AWS IoT Core 並訂閱主題，以對來自其他來源的 MQTT 訊息採取行動。如需 MQTT AWS IoT Core 實作的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [MQTT](#)。

Note

此 MQTT 訊息 IPC 服務可讓您與 交換訊息 AWS IoT Core。如需如何在元件之間交換訊息的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [PublishToIoTCore](#)

- [SubscribeToIoTCore](#)
- [範例](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來發佈和訂閱 MQTT 訊息的最小版本 AWS IoT Core。

SDK	最低版本	
AWS IoT Device SDK for Java v2	v1.2.10	
AWS IoT Device SDK for Python v2	1.5.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Authorization

若要在自訂元件中使用 AWS IoT Core MQTT 訊息，您必須定義授權政策，以允許您的元件傳送和接收主題的訊息。如需定義授權政策的資訊，請參閱 [授權元件執行 IPC 操作](#)。

AWS IoT Core MQTT 訊息的授權政策具有下列屬性。

IPC 服務識別符：`aws.greengrass.ipc.mqttproxy`

作業	Description	Resources
<code>aws.greengrass#PublishToIoTCore</code>	允許元件在您指定的 MQTT 主題 AWS IoT Core 上發佈訊息至。	主題字串，例如 <code>test/topic</code> 或 <code>*</code> 允許存取所有主題。您可以使用 MQTT 主題萬用字元 (<code>#</code> 和 <code>+</code>) 來比對多個資源。

作業	Description	Resources
<code>aws.greengrass#SubscribeToIoTCore</code>	允許元件在您指定的主題 AWS IoT Core 上訂閱來自 的訊息。	主題字串，例如 <code>test/topic</code> 或 <code>*</code> 允許存取所有主題。您可以使用 MQTT 主題萬用字元 (<code>#</code> 和 <code>+</code>) 來比對多個資源。
<code>*</code>	允許元件發佈和訂閱您指定主題的 AWS IoT Core MQTT 訊息。	主題字串，例如 <code>test/topic</code> 或 <code>*</code> 允許存取所有主題。您可以使用 MQTT 主題萬用字元 (<code>#</code> 和 <code>+</code>) 來比對多個資源。

MQTT 授權政策中的 AWS IoT Core MQTT 萬用字元

您可以在 MQTT IPC 授權政策中使用 AWS IoT Core MQTT 萬用字元。元件可以發佈和訂閱與您在授權政策中允許的主題篩選條件相符的主題。例如，如果元件的授權政策授予的存取權 `test/topic/#`，則該元件可以訂閱 `test/topic/#`，而且可以發佈和訂閱 `test/topic/filter`。

AWS IoT Core MQTT 授權政策中的配方變數

如果您使用 [Greengrass 核](#) 的 v2.6.0 或更新版本，您可以在授權政策中使用 `{iot:thingName}` 配方變數。此功能可讓您為一組核心裝置設定單一授權政策，其中每個核心裝置只能存取包含自己的名稱的主題。例如，您可以允許元件存取下列主題資源。

```
devices/{iot:thingName}/messages
```

如需詳細資訊，請參閱[配方變數](#)及[在合併更新中使用配方變數](#)。

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example 具有無限制存取的授權政策範例

下列範例授權政策允許元件發佈和訂閱所有主題。

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具有有限存取權的授權政策範例

下列範例授權政策允許元件發佈和訂閱兩個名為 `factory/1/events` 和 `factory/1/actions` 的主題。

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 一組核心裝置的範例授權政策

Important

此範例使用適用於 [Greengrass 核元件](#) v2.6.0 和更新版本的功能。Greengrass 核 v2.6.0 在元件組態中新增了對大多數 [配方變數](#) 的支援{iot:thingName}，例如。

下列範例授權政策允許元件發佈和訂閱主題，其中包含執行元件的核心裝置名稱。

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

在主題 AWS IoT Core 上將 MQTT 訊息發佈至。

當您發佈 MQTT 訊息到時 AWS IoT Core，每秒有 100 筆交易的配額。如果您超過此配額，訊息會排入佇列，以便在 Greengrass 裝置上處理。也有每秒 512 Kb 的資料配額，以及每秒 20,000 個發佈的全帳戶配額（部分為 2,000 AWS 區域）。如需中 MQTT 訊息中介裝置限制的詳細資訊 AWS IoT Core，請參閱[AWS IoT Core 訊息中介裝置和通訊協定限制和配額](#)。

如果您超過這些配額，Greengrass 裝置會將訊息發佈限制為 AWS IoT Core。訊息會存放在記憶體中的多工緩衝處理程式中。根據預設，配置給多工緩衝處理站的記憶體為 2.5 Mb。如果多工緩衝處理程式填滿，則會拒絕新訊息。您可以增加多工緩衝處理的大小。如需詳細資訊，請參閱 [Greengrass 核](#) 文件中的 [Configuration](#)。為了避免填入多工緩衝處理程式並需要增加配置的記憶體，請將發佈請求限制為每秒不超過 100 個請求。

當您的應用程式需要以更高的速率或更大的訊息傳送訊息時，請考慮使用 [串流管理員](#) 傳送訊息至 Kinesis Data Streams。串流管理員元件旨在將大量資料傳輸至 AWS 雲端。如需詳細資訊，請參閱[管理 Greengrass 核心裝置上的資料串流](#)。

請求

此操作的請求具有下列參數：

topicName (Python : topic_name)

要發佈訊息的主題。

qos

要使用的 MQTT QoS。此列舉 QoS 具有下列值：

- AT_MOST_ONCE – QoS 0。MQTT 訊息最多傳遞一次。
- AT_LEAST_ONCE – QoS 1。MQTT 訊息至少傳遞一次。

payload

(選用) 作為 Blob 的訊息承載。

使用 MQTT 5 [Greengrass](#) 核時，下列功能適用於 2.10.0 版和更新版本。當您使用 MQTT 3.1.1 時，會忽略這些功能。下表列出您必須用來存取這些功能的 AWS IoT 裝置 SDK 最低版本。

SDK	最低版本
適用於 Python 的 AWS IoT Device SDK v2	1.15.0 版
適用於 JAVA 的 AWS IoT Device SDK v2	1.13.0 版
適用於 C++ 的 AWS IoT Device SDK v2	1.24.0 版
適用於 JavaScript 的 AWS IoT Device SDK v2	1.13.0 版

payloadFormat

(選用) 訊息承載的格式。如果您未設定 payloadFormat，則會假設類型為 BYTES。列舉具有下列值：

- BYTES – 承載的內容是二進位 Blob。

- UTF8 – 承載的內容是 UTF8 字元字串。

retain

(選用) 指出是否要在發佈true時將 MQTT 保留選項設定為。

userProperties

(選用) 要傳送的應用程式特定UserProperty物件清單。UserProperty 物件的定義如下：

```
UserProperty:
  key: string
  value: string
```

messageExpiryIntervalSeconds

(選用) 訊息過期並由伺服器刪除之前的秒數。如果未設定此值，則訊息不會過期。

correlationData

(選用) 新增至請求的資訊，可用於將請求與回應建立關聯。

responseTopic

(選用) 應該用於回應訊息的主題。

contentType

(選用) 訊息內容類型的應用程式特定識別符。

回應

此操作不會在其回應中提供任何資訊。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V2)

Example範例：發佈訊息

```
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 (GreengrassCoreIPCCClientV2 ipcClientV2 =
GreengrassCoreIPCCClientV2.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範例：發佈訊息

Note

此範例假設您使用 AWS IoT Device SDK 適用於 Python v2 的 1.5.4 版或更新版本。

```

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)

```

```
ipc_client.close()
```

Java (IPC client V1)

Example範例：發佈訊息

Note

此範例使用 IPCUtils類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱[連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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範例：發佈訊息

Note

此範例假設您使用 AWS IoT Device SDK 適用於 Python v2 的 1.5.4 版或更新版本。

```
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範例：發佈訊息

```
#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範例：發佈訊息

```

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
            }

            this.ipcClient = await getIpClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }

}

export async function getIpClient(){
    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範例：發佈訊息

```
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範例：發佈訊息

```
#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範例：發佈訊息

```
#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

在主題或主題篩選條件 AWS IoT Core 上，從訂閱 MQTT 訊息。AWS IoT Greengrass 核心軟體會在元件生命週期結束時移除訂閱。

此操作是您訂閱事件訊息串流的訂閱操作。若要使用此操作，請使用處理事件訊息、錯誤和串流關閉的函數來定義串流回應處理常式。如需詳細資訊，請參閱[訂閱 IPC 事件串流](#)。

事件訊息類型：IoTCoreMessage

請求

此操作的請求具有下列參數：

topicName (Python : topic_name)

要訂閱的主題。您可以使用 MQTT 主題萬用字元 (# 和 +) 來訂閱多個主題。

qos

要使用的 MQTT QoS。此列舉 QoS 具有下列值：

- AT_MOST_ONCE – QoS 0。MQTT 訊息最多傳遞一次。
- AT_LEAST_ONCE – QoS 1。MQTT 訊息至少傳遞一次。

回應

此操作的回應包含下列資訊：

messages

MQTT 訊息的串流。此物件 IoTCoreMessage 包含下列資訊：

message

MQTT 訊息。此物件 MQTTMessage 包含下列資訊：

topicName (Python : topic_name)

發佈訊息的主題。

payload

(選用) 作為 Blob 的訊息承載。

使用 MQTT 5 [Greengrass 核時](#)，下列功能適用於 2.10.0 版和更新版本。當您使用 MQTT 3.1.1 時，會忽略這些功能。下表列出您必須用來存取這些功能的 AWS IoT 裝置 SDK 最低版本。

SDK	最低版本
適用於 Python 的 AWS IoT Device SDK v2	1.15.0 版
適用於 JAVA 的 AWS IoT Device SDK v2	1.13.0 版
適用於 C++ 的 AWS IoT Device SDK v2	1.24.0 版
適用於 JavaScript 的 AWS IoT Device SDK v2	1.13.0 版

payloadFormat

(選用) 訊息承載的格式。如果您未設定 payloadFormat，則會假設類型為 BYTES。列舉具有下列值：

- BYTES – 承載的內容是二進位 Blob。
- UTF8 – 承載的內容是 UTF8 字元字串。

retain

(選用) 指出是否要在發佈true時將 MQTT 保留選項設定為。

userProperties

(選用) 要傳送的應用程式特定UserProperty物件清單。UserProperty 物件的定義如下：

```
UserProperty:
  key: string
  value: string
```

messageExpiryIntervalSeconds

(選用) 訊息過期並由伺服器刪除之前的秒數。如果未設定此值，則訊息不會過期。

correlationData

(選用) 新增至請求的資訊，可用於將請求與回應建立關聯。

responseTopic

(選用) 應該用於回應訊息的主題。

contentType

(選用) 訊息內容類型的應用程式特定識別符。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V2)

Example 範例：訂閱訊息

```
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範例：訂閱訊息

Note

此範例假設您使用 AWS IoT Device SDK 適用於 Python v2 的 1.5.4 版或更新版本。

```
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範例：訂閱訊息

Note

此範例使用 IPCUtils類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱[連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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(GreengrassCoreIPCCClient greengrassCoreIPCCClient, String topic,
    QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
    SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
    greengrassCoreIPCCClient.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 範例：訂閱訊息

Note

此範例假設您使用 AWS IoT Device SDK 適用於 Python v2 的 1.5.4 版或更新版本。

```
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範例：訂閱訊息

```

#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範例：訂閱訊息

```
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 getIpClient();

      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範例：訂閱訊息

```

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範例：訂閱訊息

```

#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範例：訂閱訊息

```
#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);
    }
}
```

```
}  
}
```

範例

使用下列範例來了解如何在元件中使用 AWS IoT Core MQTT IPC 服務。

範例 AWS IoT Core MQTT 發佈者 (C++、IPC 用戶端 V1)

下列範例配方允許 元件發佈至所有主題。

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

```

下列 C++ 應用程式範例示範如何使用 AWS IoT Core MQTT IPC 服務來發佈訊息 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;
}

```

範例 AWS IoT Core MQTT 訂閱者 (C++、IPC 用戶端 V1)

下列範例配方允許 元件訂閱所有主題。

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

```

下列 C++ 應用程式範例示範如何使用 AWS IoT Core MQTT IPC 服務來訂閱訊息 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;
}

```

範例 AWS IoT Core MQTT 發佈者 (Rust)

下列範例配方允許 元件發佈至所有主題。

```

{
  "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"
      }
    }
  ]
}
]
}
}

```

下列範例 Rust 應用程式示範如何使用 AWS IoT Core MQTT IPC 服務來發佈訊息 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}");
}

```

範例 AWS IoT Core MQTT 訂閱者 (Rust)

下列範例配方允許 元件訂閱所有主題。

```

{
  "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"
        }
      }
    ]
  }
]
}

```

下列範例 Rust 應用程式示範如何使用 AWS IoT Core MQTT IPC 服務來訂閱訊息 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));
    }
}

```

範例 AWS IoT Core MQTT 發佈者 (C)

下列範例配方允許 元件發佈至所有主題。

```

{
  "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"
      }
    }
  ]
}
]
}
}

```

下列範例 C 應用程式示範如何使用 AWS IoT Core MQTT IPC 服務來發佈訊息 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
);
}

```

範例 AWS IoT Core MQTT 訂閱者 (C)

下列範例配方允許 元件訂閱所有主題。

```

{
  "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"
        }
      }
    ]
  }
]
}

```

下列範例 C 應用程式示範如何使用 AWS IoT Core MQTT IPC 服務來訂閱訊息 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);  
}
```

範例 AWS IoT Core MQTT 發佈者 (C++、元件 SDK)

下列範例配方允許 元件發佈至所有主題。

```
{
  "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"
          }
        }
      ]
    }
  ]
}
```

```
}

```

下列 C++ 應用程式範例示範如何使用 AWS IoT Core MQTT IPC 服務來發佈訊息 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";
}
```

範例 AWS IoT Core MQTT 訂閱者 (C++、元件 SDK)

下列範例配方允許 元件訂閱所有主題。

```
{
  "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"
          }
        }
      ]
    }
  ]
}

```

下列 C++ 應用程式範例示範如何使用 AWS IoT Core MQTT IPC 服務來訂閱訊息 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
  ) {
    // Handle the message
  }
};

```

```
    ) 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);
    }
}
```

與元件生命週期互動

使用元件生命週期 IPC 服務來：

- 更新核心裝置上的元件狀態。
- 訂閱元件狀態更新。
- 防止核心在部署期間停止元件以套用更新。

- 暫停和繼續元件程序。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [UpdateState](#)
- [SubscribeToComponentUpdates](#)
- [DeferComponentUpdate](#)
- [PauseComponent](#)
- [ResumeComponent](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來與元件生命週期互動的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	v1.2.10	
AWS IoT Device SDK for Python v2	1.5.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Authorization

若要暫停或繼續自訂元件中的其他元件，您必須定義授權政策，以允許元件管理其他元件。如需定義授權政策的資訊，請參閱 [授權元件執行 IPC 操作](#)。

元件生命週期管理的授權政策具有下列屬性。

IPC 服務識別符：aws.greengrass.ipc.lifecycle

作業	Description	Resources
aws.greengrass#PauseComponent	允許元件暫停您指定的元件。	元件名稱，或 * 允許存取所有元件。
aws.greengrass#ResumeComponent	允許元件繼續您指定的元件。	元件名稱，或 * 允許存取所有元件。
*	允許元件暫停和繼續您指定的元件。	元件名稱，或 * 允許存取所有元件。

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example範例授權政策

下列範例授權政策允許元件暫停和繼續所有元件。

```
{
  "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

更新核心裝置上的元件狀態。

請求

此操作的請求具有下列參數：

state

要設定的狀態。此列舉 LifecycleState 具有下列值：

- RUNNING
- ERRORED

回應

此操作不會在其回應中提供任何資訊。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Rust

Example 範例：更新狀態

```
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 範例：更新狀態

```
#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範例：更新狀態

```
#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

在 AWS IoT Greengrass Core 軟體更新元件之前訂閱以接收通知。通知會指定是否要在更新時重新啟動核。

只有當部署的元件更新政策指定通知元件時，核才會傳送更新通知。預設行為是通知元件。如需詳細資訊，請參閱 [建立部署](#) 和您可以在呼叫 [CreateDeployment](#) 操作時提供的 [DeploymentComponentUpdatePolicy](#) 物件。

Important

本機部署不會在更新之前通知元件。

此操作是您訂閱事件訊息串流的訂閱操作。若要使用此操作，請使用處理事件訊息、錯誤和串流關閉的函數來定義串流回應處理常式。如需詳細資訊，請參閱 [訂閱 IPC 事件串流](#)。

事件訊息類型：ComponentUpdatePolicyEvents

Tip

您可以遵循教學課程，了解如何開發有條件延遲元件更新的元件。如需詳細資訊，請參閱 [教學課程：開發可延遲元件更新的 Greengrass 元件](#)。

請求

此操作的請求沒有任何參數。

回應

此操作的回應包含下列資訊：

messages

通知訊息的串流。此物件 ComponentUpdatePolicyEvents 包含下列資訊：

`preUpdateEvent` (Python : `pre_update_event`)

(選用) 表示核心想要更新元件的事件。您可以使用 [DeferComponentUpdate](#) 操作來回應，以確認或延遲更新，直到您的元件準備好重新啟動為止。此物件 `PreComponentUpdateEvent` 包含下列資訊：

`deploymentId` (Python : `deployment_id`)

更新元件的 AWS IoT Greengrass 部署 ID。

`isGgcRestarting` (Python : `is_ggc_restarting`)

是否需要重新啟動核心，才能套用更新。

`postUpdateEvent` (Python : `post_update_event`)

(選用) 指出核心已更新元件的事件。此物件 `PostComponentUpdateEvent` 包含下列資訊：

`deploymentId` (Python : `deployment_id`)

更新元件的 AWS IoT Greengrass 部署 ID。

 Note

此功能需要 2.7.0 版或更新版本的 Greengrass 核元件。

DeferComponentUpdate

使用 確認或延遲您探索的元件更新 [SubscribeToComponentUpdates](#)。如果您的元件已準備好讓元件更新繼續，您可以指定在核心再次檢查之前等待的時間量。您也可以使用此操作來告知 nucleus 您的元件已準備好進行更新。

如果元件未回應元件更新通知，則 nucleus 會等待您在部署的元件更新政策中指定的時間量。逾時之後，核會繼續進行部署。預設元件更新逾時為 60 秒。如需詳細資訊，請參閱 [建立部署](#) 和您可以在呼叫 [CreateDeployment](#) 操作時提供的 [DeploymentComponentUpdatePolicy](#) 物件。

 Tip

您可以遵循教學課程，了解如何開發有條件延遲元件更新的元件。如需詳細資訊，請參閱 [教學課程：開發可延遲元件更新的 Greengrass 元件](#)。

請求

此操作的請求具有下列參數：

`deploymentId` (Python : `deployment_id`)

要延遲的 AWS IoT Greengrass 部署 ID。

`message`

(選用) 要延遲更新的元件名稱。

預設為發出請求的元件名稱。

`recheckAfterMs` (Python : `recheck_after_ms`)

延遲更新的時間量，以毫秒為單位。核會等待這段時間，然後傳送另一個 `PreComponentUpdateEvent` 您可以使用 探索的時間 [SubscribeToComponentUpdates](#)。

指定 0 以確認更新。這會通知 nucleus 您的元件已準備好進行更新。

預設為零毫秒，這表示 確認更新。

回應

此操作不會在其回應中提供任何資訊。

PauseComponent

此功能適用於 [Greengrass 核元件](#) 的 v2.4.0 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

在核心裝置上暫停元件的程序。若要繼續元件，請使用 [ResumeComponent](#) 操作。

您只能暫停一般元件。如果您嘗試暫停任何其他類型的元件，此操作會擲回 `InvalidRequestError`。

Note

此操作無法暫停容器化程序，例如 Docker 容器。若要暫停和繼續 Docker 容器，您可以使用 [docker pause](#) 和 [docker unpause](#) 命令。

此操作不會暫停元件相依性，或取決於您暫停之元件的元件。當您暫停屬於另一個元件相依性的元件時，請考慮此行為，因為相依元件可能會在其相依性暫停時遇到問題。

當您重新啟動或關閉暫停的元件時，例如透過部署，Greengrass 核會繼續該元件並執行其關機生命週期。如需重新啟動元件的詳細資訊，請參閱 [RestartComponent](#)。

Important

若要使用此操作，您必須定義授權政策，授予使用此操作的許可。如需詳細資訊，請參閱 [Authorization](#)。

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來暫停和繼續元件的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.4.3 版	
AWS IoT Device SDK for Python v2	1.6.2 版	
AWS IoT Device SDK 適用於 C++ v2	V1.13.1	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

請求

此操作的請求具有下列參數：

componentName (Python : component_name)

要暫停的元件名稱，必須是一般元件。如需詳細資訊，請參閱 [元件類型](#)。

回應

此操作不會在其回應中提供任何資訊。

ResumeComponent

此功能適用於 [Greengrass 核元件](#) 的 v2.4.0 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

在核心裝置上繼續元件的程序。若要暫停元件，請使用 [PauseComponent](#) 操作。

您只能繼續暫停的元件。如果您嘗試繼續未暫停的元件，此操作會擲回 `InvalidRequestError`。

Important

若要使用此操作，您必須定義授予許可的授權政策。如需詳細資訊，請參閱[Authorization](#)。

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來暫停和繼續元件的 最低版本。

SDK	最低版本
AWS IoT Device SDK for Java v2	1.4.3 版
AWS IoT Device SDK for Python v2	1.6.2 版
AWS IoT Device SDK 適用於 C++ v2	V1.13.1
AWS IoT Device SDK for JavaScript v2	v1.12.0

請求

此操作的請求具有下列參數：

componentName (Python : component_name)

要繼續的元件名稱。

回應

此操作不會在其回應中提供任何資訊。

與元件組態互動

元件組態 IPC 服務可讓您執行下列動作：

- 取得並設定元件組態參數。
- 訂閱元件組態更新。
- 在套用核心之前驗證元件組態更新。

主題

- [最低 SDK 版本](#)
- [GetConfiguration](#)
- [UpdateConfiguration](#)
- [SubscribeToConfigurationUpdate](#)
- [SubscribeToValidateConfigurationUpdates](#)
- [SendConfigurationValidityReport](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來與元件組態互動的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	v1.2.10	
AWS IoT Device SDK for Python v2	1.5.3 版	

SDK	最低版本	
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

GetConfiguration

取得核心裝置上元件的組態值。您可以指定要取得組態值的金鑰路徑。

請求

此操作的請求具有下列參數：

`componentName` (Python : `component_name`)

(選用) 元件的名稱。

預設為發出請求的元件名稱。

`keyPath` (Python : `key_path`)

組態值的金鑰路徑。指定清單，其中每個項目都是組態物件中單一層級的金鑰。例如，指定 `["mqtt", "port"]` 在下列組態port中取得 的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

若要取得元件的完整組態，請指定空白清單。

回應

此操作的回應包含下列資訊：

componentName (Python : component_name)

元件的名稱。

value

請求的組態做為 物件。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Rust

Example範例：取得組態

```
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範例：取得組態

```
#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範例：取得組態

```

#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

更新核心裝置上此元件的組態值。

請求

此操作的請求具有下列參數：

keyPath (Python : key_path)

(選用) 要更新的容器節點 (物件) 的金鑰路徑。指定清單，其中每個項目都是組態物件中單一層級的金鑰。例如，指定金鑰路徑["mqtt"]和合併值{ "port": 443 }，以在下列組態port中設定的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

金鑰路徑必須在組態中指定容器節點（物件）。如果節點不存在於元件的組態中，此操作會建立節點，並將其值設定為 中的物件valueToMerge。

預設為組態物件的根。

timestamp

目前的 Unix epoch 時間，以毫秒為單位。此操作使用此時間戳記來解析金鑰的並行更新。如果元件組態中的索引鍵的時間戳記大於請求中的時間戳記，則請求會失敗。

valueToMerge (Python : value_to_merge)

要在您在 中指定的位置合併的組態物件keyPath。如需詳細資訊，請參閱[更新元件組態](#)。

回應

此操作不會在其回應中提供任何資訊。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Rust

Example範例：更新組態

```
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範例：更新組態

```
#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範例：更新組態

```

#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

訂閱以在元件的組態更新時接收通知。當您訂閱金鑰時，當該金鑰的任何子項更新時，您會收到通知。

此操作是您訂閱事件訊息串流的訂閱操作。若要使用此操作，請使用處理事件訊息、錯誤和串流關閉的函數來定義串流回應處理常式。如需詳細資訊，請參閱[訂閱 IPC 事件串流](#)。

事件訊息類型：ConfigurationUpdateEvents

請求

此操作的請求具有下列參數：

componentName (Python : component_name)

(選用) 元件的名稱。

預設為發出請求的元件名稱。

keyPath (Python : key_path)

要訂閱之組態值的金鑰路徑。指定清單，其中每個項目都是組態物件中單一層級的金鑰。例如，指定 ["mqtt", "port"] 在下列組態port中取得 的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

若要訂閱元件組態中所有值的更新，請指定空白清單。

回應

此操作的回應包含下列資訊：

messages

通知訊息的串流。此物件 ConfigurationUpdateEvents 包含下列資訊：

configurationUpdateEvent (Python : configuration_update_event)

組態更新事件。此物件 ConfigurationUpdateEvent 包含下列資訊：

componentName (Python : component_name)

元件的名稱。

keyPath (Python : key_path)

已更新之組態值的金鑰路徑。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Rust

Example 範例：訂閱組態更新

```
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範例：訂閱組態更新

```

#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範例：訂閱組態更新

```
#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

訂閱以在此元件的組態更新之前接收通知。這可讓元件驗證其自身組態的更新。使用 [SendConfigurationValidityReport](#) 操作來告知核心組態是否有效。

Important

本機部署不會通知更新元件。

此操作是您訂閱事件訊息串流的訂閱操作。若要使用此操作，請使用處理事件訊息、錯誤和串流關閉的函數來定義串流回應處理常式。如需詳細資訊，請參閱 [訂閱 IPC 事件串流](#)。

事件訊息類型：ValidateConfigurationUpdateEvents

請求

此操作的請求沒有任何參數。

回應

此操作的回應包含下列資訊：

messages

通知訊息的串流。此物件 ValidateConfigurationUpdateEvents 包含下列資訊：

`validateConfigurationUpdateEvent`

(Python : `validate_configuration_update_event`)

組態更新事件。此物件 `ValidateConfigurationUpdateEvent` 包含下列資訊：

`deploymentId` (Python : `deployment_id`)

更新元件的 AWS IoT Greengrass 部署 ID。

`configuration`

包含新組態的物件。

SendConfigurationValidityReport

告知核心此元件的組態更新是否有效。如果您告訴 nucleus 新組態無效，則部署會失敗。使用 [SubscribeToValidateConfigurationUpdates](#) 操作來訂閱以驗證組態更新。

如果元件未回應驗證組態更新通知，則 nucleus 會等待您在部署的組態驗證政策中指定的時間量。逾時之後，核會繼續進行部署。預設元件驗證逾時為 20 秒。如需詳細資訊，請參閱 [建立部署](#) 和您可以在呼叫 [CreateDeployment](#) 操作時提供的 [DeploymentConfigurationValidationPolicy](#) 物件。

請求

此操作的請求具有下列參數：

`configurationValidityReport` (Python : `configuration_validity_report`)

此報告會告知 nucleus 組態更新是否有效。此物件 `ConfigurationValidityReport` 包含下列資訊：

`status`

有效性狀態。此列舉 `ConfigurationValidityStatus` 具有下列值：

- ACCEPTED – 組態有效，且 核可以套用到此元件。
- REJECTED – 組態無效且部署失敗。

`deploymentId` (Python : `deployment_id`)

請求組態更新的 AWS IoT Greengrass 部署 ID。

`message`

(選用) 報告組態為何無效的訊息。

回應

此操作不會在其回應中提供任何資訊。

擷取秘密值

使用 Secret Manager IPC 服務，從核心裝置上的秘密擷取秘密值。您可以使用[秘密管理員元件](#)，將加密的秘密部署到核心裝置。然後，您可以使用 IPC 操作來解密秘密，並在自訂元件中使用其值。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [GetSecretValue](#)
- [範例](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來從核心裝置上的秘密擷取秘密值的 最低版本。

SDK	最低版本
AWS IoT Device SDK for Java v2	v1.2.10
AWS IoT Device SDK for Python v2	1.5.3 版
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版
AWS IoT Device SDK for JavaScript v2	v1.12.0

Authorization

若要在自訂元件中使用秘密管理員，您必須定義授權政策，允許元件取得存放在核心裝置上的秘密值。如需定義授權政策的資訊，請參閱[授權元件執行 IPC 操作](#)。

秘密管理員的授權政策具有下列屬性。

IPC 服務識別符：`aws.greengrass.SecretManager`

作業	Description	Resources
<code>aws.greengrass#GetSecretValue</code> 或 *	允許元件取得在核心裝置上加密的秘密值。	Secrets Manager 秘密 ARN，或 * 允許存取所有秘密。

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example 範例授權政策

下列範例授權政策允許元件取得核心裝置上任何秘密的值。

Note

我們建議您在生產環境中減少授權政策的範圍，以便元件僅擷取其使用的秘密。部署元件時，您可以將 * 萬用字元變更為秘密 ARNs 清單。

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

取得您存放在核心裝置上的秘密值。

此操作類似於 Secrets Manager 操作，您可以用來取得 中的秘密值 AWS 雲端。如需詳細資訊，請參閱 AWS Secrets Manager API 參考中的 [GetSecretValue](#)。

請求

此操作的請求具有下列參數：

`refresh` (Python : `refresh`)

(選用)：是否要將請求的秘密與來自 AWS Secrets Manager 服務的最新值同步。

當設定為 `true` 時，秘密管理員會請求 AWS Secrets Manager 服務以取得指定秘密標籤的最新值，並將該值傳回為回應。否則，將會傳回儲存在本機的秘密值。

此參數無法與請求中的 `versionId` 參數搭配使用。此參數可搭配 Nucleus 2.13.0 及更高版本使用。

`secretId` (Python : `secret_id`)

要取得的秘密名稱。您可以指定 Amazon Resource Name (ARN) 或秘密的易記名稱。

`versionId` (Python : `version_id`)

(選用) 要取得的版本 ID。

您可指定為 `versionId` 或 `versionStage`。

如果您未指定 `versionId` 或 `versionStage`，則此操作預設為具有 `AWSCURRENT` 標籤的版本。

`versionStage` (Python : `version_stage`)

(選用) 要取得之版本的預備標籤。

您可指定為 `versionId` 或 `versionStage`。

如果您未指定 `versionId` 或 `versionStage`，則此操作預設為具有 `AWSCURRENT` 標籤的版本。

回應

此操作的回應包含下列資訊：

`secretId` (Python : `secret_id`)

秘密的 ID。

`versionId` (Python : `version_id`)

此版本秘密的 ID。

`versionStage` (Python : `version_stage`)

連接到此版本秘密的預備標籤清單。

`secretValue` (Python : `secret_value`)

此版本秘密的值。此物件 `SecretValue` 包含下列資訊。

`secretString` (Python : `secret_string`)

您以字串形式提供給 Secrets Manager 的受保護秘密資訊的解密部分。

`secretBinary` (Python : `secret_binary`)

(選用) 您以位元組陣列形式提供給 Secrets Manager 做為二進位資料的受保護秘密資訊的解密部分。此屬性包含二進位資料做為 base64 編碼字串。

如果您在 Secrets Manager 主控台中建立秘密，則不會使用此屬性。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V1)

Example 範例：取得秘密值

Note

此範例使用 `IPCUtils` 類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱 [連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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範例：取得秘密值

Note

此範例假設您使用 AWS IoT Device SDK 適用於 Python v2 的 1.5.4 版或更新版本。

```

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範例：取得秘密值

```

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();
```

範例

使用下列範例來了解如何在元件中使用秘密管理員 IPC 服務。

範例：列印秘密 (Python、IPC 用戶端 V1)

此範例元件會列印您部署到核心裝置的秘密值。

Important

此範例元件會列印秘密的值，因此只能與存放測試資料的秘密搭配使用。請勿使用此元件來列印存放重要資訊的秘密值。

主題

- [Recipe](#)
- [成品](#)
- [Usage](#)

Recipe

下列範例配方會定義秘密 ARN 組態參數，並允許元件取得核心裝置上任何秘密的值。

Note

我們建議您在生產環境中減少授權政策的範圍，以便元件僅擷取其使用的秘密。部署元件時，您可以將*萬用字元變更為秘密 ARNs 清單。

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}"

```

成品

下列範例 Python 應用程式示範如何使用 Secret Manager IPC 服務來取得核心裝置上的秘密值。

```

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)
```

Usage

您可以使用此範例元件搭配[秘密管理員元件](#)，在核心裝置上部署和列印秘密的值。

建立、部署和列印測試秘密

1. 使用測試資料建立 Secrets Manager 秘密。

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"}'
```

儲存秘密的 ARN，以便在下列步驟中使用。

如需詳細資訊，請參閱AWS Secrets Manager 《使用者指南》中的[建立秘密](#)。

2. 使用下列組態合併更新來部署[秘密管理員元件](#) (aws.greengrass.SecretManager)。指定您先前建立之秘密的 ARN。

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

如需詳細資訊，請參閱 [將 AWS IoT Greengrass 元件部署至裝置](#)或 [Greengrass CLI 部署命令](#)。

3. 使用下列組態合併更新，在本節中建立和部署範例元件。指定您先前建立之秘密的 ARN。

```
{
  "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"
    ]
  }
}
```

如需詳細資訊，請參閱[建立 AWS IoT Greengrass 元件](#)

4. 檢視 AWS IoT Greengrass 核心軟體日誌以驗證部署是否成功，並檢視 `com.example.PrintSecret` 元件日誌以查看列印的秘密值。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

與本機陰影互動

使用陰影 IPC 服務與裝置上的本機陰影互動。您選擇要互動的裝置可以是您的核心裝置或連線的用戶端裝置。

若要使用這些 IPC 操作，請在自訂[元件中包含影子管理員](#)元件做為相依性。然後，您可以在自訂元件中使用 IPC 操作，透過影子管理員與裝置上的本機影子互動。若要讓自訂元件對本機影子狀態的變更做出反應，您也可以使用發佈/訂閱 IPC 服務來訂閱影子事件。如需使用發佈/訂閱服務的詳細資訊，請參閱[發佈/訂閱本機訊息](#)。

Note

若要讓核心裝置與用戶端裝置影子互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱[啟用影子管理員以與用戶端裝置通訊](#)。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [GetThingShadow](#)

- [UpdateThingShadow](#)
- [DeleteThingShadow](#)
- [ListNamedShadowsForThing](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來與本機影子互動的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.4.0 版	
AWS IoT Device SDK for Python v2	v1.6.0	
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Authorization

若要在自訂元件中使用陰影 IPC 服務，您必須定義授權政策，允許您的元件與陰影互動。如需定義授權政策的資訊，請參閱 [授權元件執行 IPC 操作](#)。

陰影互動的授權政策具有下列屬性。

IPC 服務識別符：`aws.greengrass.ShadowManager`

作業	Description	Resources
<code>aws.greengrass#GetThingShadow</code>	允許元件擷取物件的陰影。	下列其中一個字串： <code>\$aws/things/<i>thingName</i> /</code>

作業	Description	Resources
		shadow/，以允許存取傳統裝置影子。 • \$aws/thin gs/ <i>thingName</i> /shadow/n ame/ <i>shadowName</i>，以允許存取具名影子。 • * 允許存取所有陰影。
aws.greengrass#UpdateThingShadow	允許元件更新物件的陰影。	下列其中一個字串： • \$aws/thin gs/ <i>thingName</i> / shadow/，以允許存取傳統裝置影子。 • \$aws/thin gs/ <i>thingName</i> /shadow/n ame/ <i>shadowName</i>，以允許存取具名影子。 • * 允許存取所有陰影。
aws.greengrass#DeleteThingShadow	允許元件刪除物件的陰影。	下列其中一個字串： • \$aws/thin gs/ <i>thingName</i> / shadow/，以允許存取傳統裝置影子 • \$aws/thin gs/ <i>thingName</i> /shadow/n ame/ <i>shadowName</i>，以允許存取具名影子 • *，以允許存取所有陰影。

作業	Description	Resources
<code>aws.greengrass#ListNamedShadowsForThing</code>	允許元件擷取物件的具名影子清單。	物件名稱字串，允許存取物件以列出其影子。 使用 * 允許存取所有物件。

IPC 服務識別符：`aws.greengrass.ipc.pubsub`

作業	Description	Resources
<code>aws.greengrass#SubscribeToTopic</code>	允許元件訂閱您指定主題的訊息。	下列其中一個主題字串： • <i>shadowTopicPrefix</i> / get/accepted • <i>shadowTopicPrefix</i> / get/rejected • <i>shadowTopicPrefix</i> / delete/accepted • <i>shadowTopicPrefix</i> / delete/rejected • <i>shadowTopicPrefix</i> / update/accepted • <i>shadowTopicPrefix</i> / update/delta • <i>shadowTopicPrefix</i> / update/rejected 主題字首的值<i>shadowTopicPrefix</i> 取決於影子的類型： • 傳統陰影：<code>\$aws/things/ <i>thingName</i> /shadow</code>

作業	Description	Resources
		命名影子：<code>\$aws/things/<i>thingName</i>/shadow/<i>n</i>ame/<i>shadowName</i></code> 使用 * 允許存取所有主題。 在 Greengrass nucleus v2.6.0 和更新版本中，您可以訂閱包含 MQTT 主題萬用字元 (# 和 +) 的主題。此主題字串支援 MQTT 主題萬用字元做為文字字元。例如，如果元件的授權政策授予的存取權 <code>test/topic/#</code>，則該元件可以訂閱 <code>test/topic/#</code>，但無法訂閱 <code>test/topic/filter</code>。

本機影子授權政策中的配方變數

如果您使用 [Greengrass 核](#) 的 v2.6.0 或更新版本，並將 Greengrass 核的 [interpolateComponentConfiguration](#) 組態選項設定為 `true`，您可以在授權政策中使用 `{iot:thingName}` [配方變數](#)。此功能可讓您為一組核心裝置設定單一授權政策，其中每個核心裝置只能存取自己的影子。例如，您可以允許元件存取陰影 IPC 操作的下列資源。

```
$aws/things/{iot:thingName}/shadow/
```

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example 範例：允許一組核心裝置與本機影子互動

Important

此範例使用適用於 [Greengrass 核元件](#) v2.6.0 和更新版本的功能。Greengrass 核 v2.6.0 在元件組態中新增了對大多數[配方變數](#)的支援{iot:thingName}，例如。若要啟用此功能，請將 Greengrass 核的 [interpolateComponentConfiguration](#) 組態選項設定為 true。如需適用於所有 Greengrass 核版本的範例，請參閱[單一核心裝置的範例授權政策](#)。

下列範例授權政策允許 元件與傳統裝置影子和執行元件myNamedShadow之核心裝置的具名影子com.example.MyShadowInteractionComponent互動。此政策也允許此元件接收有關這些影子之本機主題的訊息。

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 範例：允許一組核心裝置與用戶端裝置影子互動

Important

此功能需要 [Greengrass 核](#) v2.6.0 或更新版本、[陰影管理員](#) v2.2.0 或更新版本，以及 [MQTT 橋接器](#) v2.2.0 或更新版本。您必須設定 MQTT 橋接器，[讓陰影管理員能夠與用戶端裝置通訊](#)。

下列範例授權政策允許 元件與名稱開頭為 的用戶端裝置的所有裝置影子 `com.example.MyShadowInteractionComponent` 互動 `MyClientDevice`。

Note

若要讓核心裝置與用戶端裝置影子互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱[啟用影子管理員以與用戶端裝置通訊](#)。

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範例：允許單一核心裝置與本機影子互動

下列範例授權政策允許 元件與傳統裝置影子和裝置 myNamedShadow 的具名影子com.example.MyShadowInteractionComponent互動MyThingName。此政策也允許此元件接收有關這些影子之本機主題的訊息。

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範例：允許一組核心裝置對本機陰影狀態變更做出反應

Important

此範例使用適用於 [Greengrass 核元件](#) v2.6.0 和更新版本的功能。Greengrass 核 v2.6.0 在元件組態中新增了對大多數[配方變數](#)的支援{iot:thingName}，例如 `iot:thingName`。若要啟用此功能，請將 Greengrass 核的 [interpolateComponentConfiguration](#) 組態選項設定為 `true`。如需適用於所有 Greengrass 核版本的範例，請參閱[單一核心裝置的範例授權政策](#)。

下列範例存取控制政策允許 自訂 `com.example.MyShadowReactiveComponent` 接收傳統裝置影子 /update/delta 主題的訊息，以及 `myNamedShadow` 每個核心裝置上執行元件的具名影子。

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範例：允許單一核心裝置對本機陰影狀態變更做出反應

下列範例存取控制政策允許 自訂com.example.MyShadowReactiveComponent接收傳統裝置影子/update/delta主題的訊息，以及裝置 myNamedShadow的具名影子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

取得指定物件的陰影。

請求

此操作的請求具有下列參數：

`thingName` (Python : `thing_name`)

物件名稱。

類型：`string`

`shadowName` (Python : `shadow_name`)

影子的名稱。若要指定物件的傳統陰影，請將此參數設定為空字串 ("")。

Warning

AWS IoT Greengrass 服務使用 `AWSThingsShadowV2Deployment` 具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

類型：string

回應

此操作的回應包含下列資訊：

payload

Blob 形式的回應狀態文件。

類型：object 包含下列資訊：

state

狀態資訊。

此物件包含下列資訊。

desired

在裝置中請求更新的狀態屬性和值。

類型：map 鍵/值對

reported

裝置報告的狀態屬性和值。

類型：map 鍵/值對

delta

所需和報告的狀態屬性和值之間的差異。只有在 desired 和 reported 狀態不同時，才會顯示此屬性。

類型：map 鍵/值對

metadata

desired 和 reported 區段中每個屬性的時間戳記，讓您可以判斷狀態的更新時間。

類型：string

timestamp

產生回應的 epoch 日期和時間。

類型：integer

clientToken (Python：clientToken)

用於比對請求和對應回應的字符

類型：string

version

本機影子文件的版本。

類型：integer

錯誤

此操作可能會傳回下列錯誤。

`InvalidArgumentsError`

本機影子服務無法驗證請求參數。如果請求包含格式不正確的 JSON 或不支援的字元，就會發生這種情況。

`ResourceNotFoundError`

找不到請求的本機影子文件。

`ServiceError`

發生內部服務錯誤，或對 IPC 服務的請求數量超過影子管理員元件中 `maxLocalRequestsPerSecondPerThing` 和 `maxTotalLocalRequestsRate` 組態參數中指定的限制。

`UnauthorizedError`

元件的授權政策不包含此操作所需的許可。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V1)

Example範例：取得物件影子

Note

此範例使用 `IPCUtils` 類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱[連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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(GreengrassCoreIPCCClient greengrassCoreIPCCClient, 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範例：取得物件影子

```

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範例：取得物件影子

```
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 cases
    throw err
  }
}

const startScript = new GetThingShadow();
```

UpdateThingShadow

更新指定物件的影子。如果影子不存在，則會建立一個影子。

請求

此操作的請求具有下列參數：

thingName (Python : thing_name)

物件名稱。

類型 : string

shadowName (Python : shadow_name)

影子的名稱。若要指定物件的傳統陰影，請將此參數設定為空字串 ("")。

 Warning

AWS IoT Greengrass 服務使用AWSManagedGreengrassV2Deployment具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

類型：string

payload

Blob 形式的請求狀態文件。

類型：object包含下列資訊：

state

要更新的狀態資訊。此 IPC 操作只會影響指定的欄位。

此物件包含下列資訊。一般而言，您會在相同的請求中使用 desired 屬性或 reported 屬性，但不會同時使用兩者。

desired

在裝置中請求更新的狀態屬性和值。

類型：map鍵/值對

reported

裝置報告的狀態屬性和值。

類型：map鍵/值對

clientToken (Python：client_token)

(選用) 用於比對請求的字符，以及用戶端字符對應的回應。

類型：string

version

(選用) 要更新的本機影子文件版本。只有當指定的版本符合其擁有的最新版本時，陰影服務才會處理更新。

類型：integer

回應

此操作的回應包含下列資訊：

payload

Blob 形式的回應狀態文件。

類型：object 包含下列資訊：

state

狀態資訊。

此物件包含下列資訊。

desired

在裝置中請求更新的狀態屬性和值。

類型：map 鍵/值對

reported

裝置報告的狀態屬性和值。

類型：map 鍵/值對

delta

裝置報告的狀態屬性和值。

類型：map 鍵/值對

metadata

desired 和 reported 區段中每個屬性的時間戳記，讓您可以判斷狀態的更新時間。

類型：string

timestamp

產生回應的 epoch 日期和時間。

類型：integer

clientToken (Python：client_token)

用於比對請求和對應回應的字符。

類型：string

version

更新完成後的本機影子文件版本。

類型：integer

錯誤

此操作可能會傳回下列錯誤。

ConflictError

本機影子服務在更新操作期間遇到版本衝突。當請求承載中的版本與最新可用本機影子文件中的版本不相符時，就會發生這種情況。

InvalidArgumentsError

本機影子服務無法驗證請求參數。如果請求包含格式不正確的 JSON 或不支援的字元，就會發生這種情況。

有效的 payload 具有下列屬性：

- state 節點存在，且是包含 desired 或 reported 狀態資訊的物件。
- desired 和 reported 節點是物件或 null。這些物件中至少有一個必須包含有效的狀態資訊。
- desired 和 reported 物件的深度不能超過八個節點。
- clientToken 值的長度不能超過 64 個字元。
- version 值必須為 1 或更高。

ServiceError

發生內部服務錯誤，或對 IPC 服務的請求數量超過影子管理員元件中 maxLocalRequestsPerSecondPerThing 和 maxTotalLocalRequestsRate 組態參數中指定的限制。

UnauthorizedError

元件的授權政策不包含此操作所需的許可。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V1)

Example 範例：更新物件影子

Note

此範例使用 `IPCUtils` 類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱[連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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(GreengrassCoreIPCCClient greengrassCoreIPCCClient, String
thingName, String shadowName, byte[] shadowPayload) {
    UpdateThingShadowRequest updateThingShadowRequest = new
UpdateThingShadowRequest();
    updateThingShadowRequest.setThingName(thingName);
    updateThingShadowRequest.setShadowName(shadowName);
    updateThingShadowRequest.setPayload(shadowPayload);
    return greengrassCoreIPCCClient.updateThingShadow(updateThingShadowRequest,
        Optional.empty());
}
}

```

Python (IPC client V1)

Example範例：更新物件影子

```
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範例：更新物件影子

```
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

刪除特定物件的影子。

從影子管理員 v2.0.4 開始，刪除影子會增加版本編號。例如，當您刪除MyThingShadow第 1 版的影子時，已刪除影子的版本為 2。如果您接著重新建立名稱為 的影子MyThingShadow，則該影子的版本為 3。

請求

此操作的請求具有下列參數：

thingName (Python : thing_name)

物件名稱。

類型 : string

shadowName (Python : shadow_name)

影子的名稱。若要指定物件的傳統陰影，請將此參數設定為空字串 ("")。

 Warning

AWS IoT Greengrass 服務使用AWSManagedGreengrassV2Deployment具名影子來管理以個別核心裝置為目標的部署。此具名影子會保留供 AWS IoT Greengrass 服務使用。請勿更新或刪除此具名影子。

類型 : string

回應

此操作的回應包含下列資訊：

payload

空的回應狀態文件。

錯誤

此操作可能會傳回下列錯誤。

InvalidArgumentsError

本機影子服務無法驗證請求參數。如果請求包含格式不正確的 JSON 或不支援的字元，就會發生這種情況。

ResourceNotFoundError

找不到請求的本機影子文件。

ServiceError

發生內部服務錯誤，或對 IPC 服務的請求數量超過影子管理員元件中 maxLocalRequestsPerSecondPerThing和 maxTotalLocalRequestsRate組態參數中指定的限制。

UnauthorizedError

元件的授權政策不包含此操作所需的許可。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V1)

Example 範例：刪除物件影子

Note

此範例使用 `IPCUtils` 類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱[連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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範例：刪除物件影子

```

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範例：刪除物件影子

```

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 getIpClient();
    } 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 getIpClient() {
  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

列出指定物件的具名影子。

請求

此操作的請求具有下列參數：

thingName (Python : thing_name)

物件名稱。

類型 : string

pageSize (Python : page_size)

(選用) 每次呼叫中要傳回的陰影名稱數量。

類型 : integer

預設 : 25

上限 : 100

nextToken (Python : next_token)

(選用) 擷取下一組結果的字符。此值會在分頁結果上傳回，並在傳回下一頁的呼叫中使用。

類型：string

回應

此操作的回應包含下列資訊：

results

陰影名稱清單。

類型：array

timestamp

(選用) 產生回應的日期和時間。

類型：integer

nextToken (Python：next_token)

(選用) 在分頁請求中使用的字符值，用於擷取序列中的下一頁。當不再傳回陰影名稱時，此字符不存在。

類型：string

Note

如果請求的頁面大小完全符合回應中的陰影名稱數量，則表示存在此字符；不過，使用時，它會傳回空清單。

錯誤

此操作可能會傳回下列錯誤。

InvalidArgumentsError

本機影子服務無法驗證請求參數。如果請求包含格式不正確的 JSON 或不支援的字元，就會發生這種情況。

ResourceNotFoundError

找不到請求的本機影子文件。

ServiceError

發生內部服務錯誤，或對 IPC 服務的請求數量超過影子管理員元件中 `maxLocalRequestsPerSecondPerThing` 和 `maxTotalLocalRequestsRate` 組態參數中指定的限制。

UnauthorizedError

元件的授權政策不包含此操作所需的許可。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Java (IPC client V1)

Example 範例：列出物件的命名影子

Note

此範例使用 `IPCUtils` 類別來建立與 AWS IoT Greengrass Core IPC 服務的連線。如需詳細資訊，請參閱[連線至 AWS IoT Greengrass Core IPC 服務](#)。

```
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範例：列出物件的命名影子

```

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範例：列出物件的命名影子

```

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();
```

管理本機部署和元件

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.6.0 和更新版本。

使用 Greengrass CLI IPC 服務來管理核心裝置上的本機部署和 Greengrass 元件。

若要使用這些 IPC 操作，請在自訂 [元件中包含 2.6.0 版或更新版本的 Greengrass CLI](#) 元件做為相依性。然後，您可以在自訂元件中使用 IPC 操作來執行下列動作：

- 建立本機部署以修改和設定核心裝置上的 Greengrass 元件。
- 在核心裝置上重新啟動和停止 Greengrass 元件。
- 產生可用來登入 [本機偵錯主控台](#) 的密碼。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [CreateLocalDeployment](#)
- [ListLocalDeployments](#)
- [GetLocalDeploymentStatus](#)

- [ListComponents](#)
- [GetComponentDetails](#)
- [RestartComponent](#)
- [StopComponent](#)
- [CreateDebugPassword](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來與 Greengrass CLI IPC 服務互動的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	v1.2.10	
AWS IoT Device SDK for Python v2	1.5.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.17.0 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Authorization

若要在自訂元件中使用 Greengrass CLI IPC 服務，您必須定義授權政策，允許您的元件管理本機部署和元件。如需定義授權政策的資訊，請參閱 [授權元件執行 IPC 操作](#)。

Greengrass CLI 的授權政策具有下列屬性。

IPC 服務識別符：`aws.greengrass.Cli`

作業	Description	Resources
<code>aws.greengrass#CreateLocalDeployment</code>	允許元件在核心裝置上建立本機部署。	*

作業	Description	Resources
<code>aws.greengrass#ListLocalDeployments</code>	允許元件列出核心裝置上的本機部署。	*
<code>aws.greengrass#GetLocalDeploymentStatus</code>	允許元件取得核心裝置上本機部署的狀態。	本機部署 ID，或 * 允許存取所有本機部署。
<code>aws.greengrass#ListComponents</code>	允許元件列出核心裝置上的元件。	*
<code>aws.greengrass#GetComponentDetails</code>	允許元件取得核心裝置上元件的詳細資訊。	元件名稱，例如 <code>com.example.HelloWorld</code> 或 * 允許存取所有元件。
<code>aws.greengrass#RestartComponent</code>	允許元件重新啟動核心裝置上的元件。	元件名稱，例如 <code>com.example.HelloWorld</code> 或 * 允許存取所有元件。
<code>aws.greengrass#StopComponent</code>	允許元件停止核心裝置上的元件。	元件名稱，例如 <code>com.example.HelloWorld</code> 或 * 允許存取所有元件。
<code>aws.greengrass#CreateDebugPassword</code>	允許元件產生密碼，以用來登入 本機偵錯主控台元件 。	*

Example範例授權政策

下列範例授權政策允許元件建立本機部署、檢視所有本機部署和元件，以及重新啟動和停止名為 `com.example.HelloWorld` 的元件。

```
{
  "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

使用指定的元件配方、成品和執行時間引數建立或更新本機部署。

此操作提供與 Greengrass CLI 中的 [部署建立命令](#) 相同的功能。

請求

此操作的請求具有下列參數：

`recipeDirectoryPath` (Python : `recipe_directory_path`)

(選用) 包含元件配方檔案之資料夾的絕對路徑。

artifactDirectoryPath (Python : artifact_directory_path)

(選用) 資料夾的絕對路徑，其中包含要包含在部署中的成品檔案。成品資料夾必須包含下列資料夾結構：

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

rootComponentVersionsToAdd (Python : root_component_versions_to_add)

(選用) 要在核心裝置上安裝的元件版本。此物件 ComponentToVersionMap 是包含下列鍵值對的映射：

key

元件的名稱。

value

元件的版本。

rootComponentsToRemove (Python : root_components_to_remove)

(選用) 從核心裝置解除安裝的元件。指定清單，其中每個項目都是元件的名稱。

componentToConfiguration (Python : component_to_configuration)

(選用) 部署中每個元件的組態更新。此物件 ComponentToConfiguration 是包含下列鍵值對的映射：

key

元件的名稱。

value

元件的組態更新 JSON 物件。JSON 物件必須具有下列格式。

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

```
}
```

如需組態更新的詳細資訊，請參閱 [更新元件組態](#)。

`componentToRunWithInfo` (Python : `component_to_run_with_info`)

(選用) 部署中每個元件的執行時間組態。此組態包含擁有每個元件程序的系統使用者，以及要套用至每個元件的系統限制。此物件 `ComponentToRunWithInfo` 是包含下列鍵值對的映射：

key

元件的名稱。

value

元件的執行時間組態。如果您省略執行時間組態參數，則 AWS IoT Greengrass Core 軟體會使用您在 [Greengrass 核](#) 上設定的預設值。此物件 `RunWithInfo` 包含下列資訊：

`posixUser` (Python : `posix_user`)

(選用) POSIX 系統使用者和選用的群組，用於在 Linux 核心裝置上執行此元件。如果指定，使用者和群組必須存在於每個 Linux 核心裝置上。以下列格式指定使用者和群組，並以冒號 (:) 分隔：`user:group`。群組為選用項目。如果您未指定群組，則 AWS IoT Greengrass Core 軟體會使用使用者的主要群組。如需詳細資訊，請參閱 [設定執行元件的使用者](#)。

`windowsUser` (Python : `windows_user`)

(選用) 用來在 Windows 核心裝置上執行此元件的 Windows 使用者。使用者必須存在於每個 Windows 核心裝置上，且其名稱和密碼必須存放在 LocalSystem 帳戶的 Credentials Manager 執行個體中。如需詳細資訊，請參閱 [設定執行元件的使用者](#)。

`systemResourceLimits` (Python : `system_resource_limits`)

(選用) 要套用至此元件程序的系統資源限制。您可以將系統資源限制套用至一般和非容器 Lambda 元件。如需詳細資訊，請參閱 [設定元件的系統資源限制](#)。

AWS IoT Greengrass 目前不支援 Windows 核心裝置上的此功能。

此物件 `SystemResourceLimits` 包含下列資訊：

`cpus`

(選用) 此元件程序可在核心裝置上使用的 CPU 時間上限。核心裝置的 CPU 總時間等於裝置的 CPU 核心數。例如，在具有 4 個 CPU 核心的核心裝置上，您可以將此值設定

為 2，將此元件的程序限制為每個 CPU 核心的 50% 用量。在具有 1 個 CPU 核心的裝置上，您可以將此值設定為 0.25，將此元件的程序限制為 CPU 的 25% 用量。如果您將此值設定為大於 CPU 核心數量的數字，則 AWS IoT Greengrass Core 軟體不會限制元件的 CPU 使用量。

memory

(選用) 此元件程序可在核心裝置上使用的 RAM 數量上限 (以 KB 為單位)。

groupName (Python : group_name)

(選用) 此部署的目標物件群組名稱。

回應

此操作的回應包含下列資訊：

deploymentId (Python : deployment_id)

請求建立的本機部署 ID。

ListLocalDeployments

取得過去 10 個本機部署的狀態。

此操作提供與 Greengrass CLI 中的[部署清單命令](#)相同的功能。

請求

此操作的請求沒有任何參數。

回應

此操作的回應包含下列資訊：

localDeployments (Python : local_deployments)

本機部署的清單。此清單中的每個物件都是 LocalDeployment 物件，其中包含下列資訊：

deploymentId (Python : deployment_id)

本機部署的 ID。

status

本機部署的狀態。此列舉 DeploymentStatus 具有下列值：

- QUEUED
- IN_PROGRESS
- SUCCEEDED
- FAILED

GetLocalDeploymentStatus

取得本機部署的狀態。

此操作提供與 Greengrass CLI 中的[部署狀態命令](#)相同的功能。

請求

此操作的請求具有下列參數：

deploymentId (Python : deployment_id)

要取得的本機部署 ID。

回應

此操作的回應包含下列資訊：

deployment

本機部署。此物件 LocalDeployment 包含下列資訊：

deploymentId (Python : deployment_id)

本機部署的 ID。

status

本機部署的狀態。此列舉 DeploymentStatus 具有下列值：

- QUEUED
- IN_PROGRESS
- SUCCEEDED

- FAILED

ListComponents

取得核心裝置上每個根元件的名稱、版本、狀態和組態。根元件是您在部署中指定的元件。此回應不包含安裝為其他元件相依性的元件。

此操作提供與 Greengrass CLI 中的 [元件清單命令](#) 相同的功能。

請求

此操作的請求沒有任何參數。

回應

此操作的回應包含下列資訊：

components

核心裝置上的根元件清單。此清單中的每個物件都是 ComponentDetails 物件，其中包含下列資訊：

componentName (Python : component_name)

元件的名稱。

version

元件的版本。

state

元件的狀態。此狀態可以是下列其中一項：

- BROKEN
- ERRORED
- FINISHED
- INSTALLED
- NEW
- RUNNING
- STARTING

- STOPPING
configuration

元件的組態做為 JSON 物件。

GetComponentDetails

取得核心裝置上元件的版本、狀態和組態。

此操作提供與 Greengrass CLI 中的[元件詳細資訊命令](#)相同的功能。

請求

此操作的請求具有下列參數：

componentName (Python : component_name)

要取得的元件名稱。

回應

此操作的回應包含下列資訊：

componentDetails (Python : component_details)

元件的詳細資訊。此物件 ComponentDetails 包含下列資訊：

componentName (Python : component_name)

元件的名稱。

version

元件的版本。

state

元件的狀態。此狀態可以是下列其中一項：

- BROKEN
- ERRORED
- FINISHED

- INSTALLED
- NEW
- RUNNING
- STARTING
- STOPPING

configuration

元件的組態做為 JSON 物件。

RestartComponent

重新啟動核心裝置上的元件。

Note

雖然您可以重新啟動任何元件，但我們建議您僅重新啟動[一般元件](#)。

此操作提供與 Greengrass CLI 中的[元件重新啟動命令](#)相同的功能。

請求

此操作的請求具有下列參數：

componentName (Python : component_name)

元件的名稱。

回應

此操作的回應包含下列資訊：

restartStatus (Python : restart_status)

重新啟動請求的狀態。請求狀態可以是下列其中一項：

- SUCCEEDED
- FAILED

message

說明如果請求失敗，為何元件無法重新啟動的訊息。

範例

下列範例示範如何在自訂元件程式碼中呼叫此操作。

Rust

Example範例：重新啟動元件

```
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範例：重新啟動元件

```
#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範例：重新啟動元件

```

#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

在核心裝置上停止元件的程序。

Note

雖然您可以停止任何元件，但我們建議您僅停止 [一般元件](#)。

此操作提供與 Greengrass CLI 中的 [元件停止命令](#) 相同的功能。

請求

此操作的請求具有下列參數：

componentName (Python : component_name)

元件的名稱。

回應

此操作的回應包含下列資訊：

stopStatus (Python : stop_status)

停止請求的狀態。請求狀態可以是下列其中一項：

- SUCCEEDED
- FAILED

message

如果請求失敗，說明元件為何無法停止的訊息。

CreateDebugPassword

產生隨機密碼，您可以用來登入[本機偵錯主控台元件](#)。密碼會在產生後 8 小時過期。

此操作提供與 Greengrass CLI 中的 [get-debug-password 命令](#) 相同的功能。

請求

此操作的請求沒有任何參數。

回應

此操作的回應包含下列資訊：

username

用於登入的使用者名稱。

password

用來登入的密碼。

passwordExpiration (Python : password_expiration)

密碼過期的時間。

certificateSHA256Hash (Python : certificate_sha256_hash)

啟用 HTTPS 時，本機偵錯主控台使用的自我簽署憑證的 SHA-256 指紋。當您開啟本機偵錯主控台時，請使用此指紋來驗證憑證是否合法且連線安全。

certificateSHA1Hash (Python : certificate_sha1_hash)

啟用 HTTPS 時，本機偵錯主控台使用的自我簽署憑證的 SHA-1 指紋。當您開啟本機偵錯主控台時，請使用此指紋來驗證憑證是否合法且連線安全。

驗證和授權用戶端裝置

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.6.0 和更新版本。

使用用戶端裝置驗證 IPC 服務來開發自訂本機代理程式元件，例如用戶端裝置等本機 IoT 裝置可以連線。

若要使用這些 IPC 操作，請在自訂[元件中包含 2.2.0 版或更新版本的用戶端裝置身分驗證](#)元件做為相依性。然後，您可以在自訂元件中使用 IPC 操作來執行下列動作：

- 驗證連接至核心裝置的用戶端裝置身分。
- 為用戶端裝置建立工作階段以連線至核心裝置。
- 驗證用戶端裝置是否具有執行 動作的許可。
- 在核心裝置的伺服器憑證輪換時接收通知。

主題

- [最低 SDK 版本](#)
- [Authorization](#)
- [VerifyClientDeviceIdentity](#)
- [GetClientDeviceAuthToken](#)
- [AuthorizeClientDeviceAction](#)
- [SubscribeToCertificateUpdates](#)

最低 SDK 版本

下表列出 AWS IoT Device SDK 您必須用來與用戶端裝置驗證 IPC 服務互動的 最低版本。

SDK	最低版本	
AWS IoT Device SDK for Java v2	1.9.3 版	
AWS IoT Device SDK for Python v2	1.11.3 版	
AWS IoT Device SDK 適用於 C++ v2	1.18.3 版	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Authorization

若要在自訂元件中使用用戶端裝置驗證 IPC 服務，您必須定義授權政策，以允許您的元件執行這些操作。如需定義授權政策的資訊，請參閱 [授權元件執行 IPC 操作](#)。

用戶端裝置身分驗證和授權的授權政策具有下列屬性。

IPC 服務識別符：`aws.greengrass.clientdevices.Auth`

作業	Description	Resources
<code>aws.greengrass#VerifyClientDeviceIdentity</code>	允許元件驗證用戶端裝置的身分。	*
<code>aws.greengrass#GetClientDeviceAuthToken</code>	允許元件驗證用戶端裝置的登入資料，並為該用戶端裝置建立工作階段。	*
<code>aws.greengrass#AuthorizeClientDeviceAction</code>	允許元件驗證用戶端裝置是否具有執行 動作的許可。	*
<code>aws.greengrass#SubscribeToCertificateUpdates</code>	允許元件在核心裝置的伺服器憑證輪換時接收通知。	*
*	允許元件執行所有用戶端裝置驗證 IPC 服務操作。	*

授權政策範例

您可以參考下列授權政策範例，協助您設定元件的授權政策。

Example範例授權政策

下列範例授權政策允許元件執行所有用戶端裝置驗證 IPC 操作。

```
{
```

```

"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

驗證用戶端裝置的身分。此操作會驗證用戶端裝置是否為有效 AWS IoT 物件。

請求

此操作的請求具有下列參數：

`credential`

用戶端裝置的登入資料。此物件 `ClientDeviceCredential` 包含下列資訊：

`clientDeviceCertificate` (Python : `client_device_certificate`)

用戶端裝置的 X.509 裝置憑證。

回應

此操作的回應包含下列資訊：

`isValidClientDevice` (Python : `is_valid_client_device`)

用戶端裝置的身分是否有效。

GetClientDeviceAuthToken

驗證用戶端裝置的登入資料，並為用戶端裝置建立工作階段。此操作會傳回工作階段字符，您可以在後續請求中使用它來[授權用戶端裝置動作](#)。

若要成功連接用戶端裝置，[用戶端裝置身分驗證元件](#)必須授予用戶端裝置使用的用戶端 ID `mqtt:connect` 許可。

請求

此操作的請求具有下列參數：

`credential`

用戶端裝置的登入資料。此物件 `CredentialDocument` 包含下列資訊：

`mqttCredential` (Python : `mqtt_credential`)

用戶端裝置的 MQTT 登入資料。指定用戶端裝置用來連線的用戶端 ID 和憑證。此物件 `MQTTCredential` 包含下列資訊：

`clientId` (Python : `client_id`)

用於連線的用戶端 ID。

`certificatePem` (Python : `certificate_pem`)

用於連線的 X.509 裝置憑證。

`username`

Note

此屬性目前未使用。

`password`

Note

此屬性目前未使用。

回應

此操作的回應包含下列資訊：

`clientDeviceAuthToken` (Python : `client_device_auth_token`)

用戶端裝置的工作階段字符。您可以在後續請求中使用此工作階段字符來授權此用戶端裝置的動作。

AuthorizeClientDeviceAction

驗證用戶端裝置是否具有對資源執行動作的許可。用戶端裝置授權政策指定用戶端裝置在連線至核心裝置時可執行的許可。您可以在設定用戶端裝置[身分驗證元件時定義用戶端裝置](#)授權政策。

請求

此操作的請求具有下列參數：

`clientDeviceAuthToken` (Python : `client_device_auth_token`)

用戶端裝置的工作階段字符。

`operation`

要授權的操作。

`resource`

用戶端裝置執行操作的資源。

回應

此操作的回應包含下列資訊：

`isAuthorized` (Python : `is_authorized`)

用戶端裝置是否獲授權在資源上執行 操作。

SubscribeToCertificateUpdates

訂閱以在每次輪換時接收核心裝置的新伺服器憑證。當伺服器憑證輪換時，代理程式必須使用新的伺服器憑證重新載入。

根據預設，[用戶端裝置身分驗證元件](#)會每 7 天輪換一次伺服器憑證。您可以將輪換間隔設定為 2 到 10 天。

此操作是您訂閱事件訊息串流的訂閱操作。若要使用此操作，請使用處理事件訊息、錯誤和串流關閉的函數來定義串流回應處理常式。如需詳細資訊，請參閱[訂閱 IPC 事件串流](#)。

事件訊息類型：CertificateUpdateEvent

請求

此操作的請求具有下列參數：

certificateOptions (Python : certificate_options)

要訂閱的憑證更新類型。此物件 CertificateOptions 包含下列資訊：

certificateType (Python : certificate_type)

要訂閱的憑證更新類型。選擇下列選項：

- SERVER

回應

此操作的回應包含下列資訊：

messages

訊息串流。此物件 CertificateUpdateEvent 包含下列資訊：

certificateUpdate (Python : certificate_update)

新憑證的相關資訊。此物件 CertificateUpdate 包含下列資訊：

certificate

憑證。

privateKey (Python : private_key)

憑證的私有金鑰。

publicKey (Python : public_key)

憑證的公有金鑰。

caCertificates (Python : ca_certificates)

憑證的 CA 憑證鏈中的憑證授權機構 (CA) 憑證清單。

與本機 IoT 裝置互動

用戶端裝置是本機 IoT 裝置，可透過 MQTT 連線至 Greengrass 核心裝置並與之通訊。您可以將用戶端裝置連接到核心裝置，以執行下列動作：

- 在 Greengrass 元件中與 MQTT 訊息互動。
- 在用戶端裝置與 之間轉送訊息和資料 AWS IoT Core。
- 在 Greengrass 元件中與用戶端裝置陰影互動。
- 與 同步用戶端裝置影子 AWS IoT Core。
- 將 IPv6 用於本機傳訊。

若要連線至核心裝置，用戶端裝置可以使用雲端探索。用戶端裝置會連線至 AWS IoT Greengrass 雲端服務，以擷取其可連線之核心裝置的相關資訊。然後，他們可以連接到核心裝置來處理其訊息，並與 AWS IoT Core 雲端服務同步其資料。

您可以遵循教學課程，逐步說明如何設定核心裝置來與 AWS IoT 物件連線和通訊。本教學課程也會探索如何開發與用戶端裝置互動的自訂 Greengrass 元件。如需詳細資訊，請參閱[教學課程：透過 MQTT 與本機 IoT 裝置互動](#)。

主題

- [AWS提供的用戶端裝置元件](#)
- [將用戶端裝置連接至核心裝置](#)
- [在用戶端裝置與 之間轉送 MQTT 訊息 AWS IoT Core](#)
- [與元件中的用戶端裝置互動](#)
- [與用戶端裝置影子互動和同步](#)
- [將 IPv6 用於本機傳訊](#)
- [對用戶端裝置進行故障診斷](#)

AWS提供的用戶端裝置元件

AWS IoT Greengrass 提供下列可部署至核心裝置的公有元件。這些元件可讓用戶端裝置與核心裝置連線和通訊。

Note

數個 AWS 提供的元件取決於 Greengrass 核的特定次要版本。由於此相依性，當您將 Greengrass 核更新為新的次要版本時，您需要更新這些元件。如需每個元件所依賴之特定核版本的相關資訊，請參閱對應的元件主題。如需更新核的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

當元件同時具有通用和 Lambda 的元件類型時，元件的目前版本是通用類型，而元件的先前版本是 Lambda 類型。

元件	描述	元件類型	支援的作業系統	開放原始碼
用戶端裝置身分驗證	讓稱為用戶端裝置的本機 IoT 裝置連接到核心裝置。	外掛程式	Linux、Windows	是
IP 偵測器	向報告 MQTT 代理程式連線資訊 AWS IoT Greengrass，以使用戶端裝置可以探索如何連線。	外掛程式	Linux、Windows	是
MQTT 橋接器	在用戶端裝置、本機 AWS IoT Greengrass 發佈/訂閱和之間轉送 MQTT 訊息 AWS IoT Core。	外掛程式	Linux、Windows	是
MQTT 3.1.1 代理程式 (Moquette)	執行 MQTT 3.1.1 代理程	外掛程式	Linux、Windows	是

元件	描述	元件類型	支援的作業系統	開放原始碼
	式，以處理用戶端裝置與核心裝置之間的訊息。			
MQTT 5 代理程式 (EMQX)	執行 MQTT 5 代理程式，處理用戶端裝置和核心裝置之間的訊息。	一般	Linux、Windows	否
Shadow 管理員	啟用與核心裝置上的陰影互動。它管理影子文件儲存，以及本機影子狀態與 AWS IoT Device Shadow 服務的同步。	外掛程式	Linux、Windows	是

將用戶端裝置連接至核心裝置

您可以設定雲端探索，將用戶端裝置連線至核心裝置。當您設定雲端探索時，用戶端裝置可以連線至 AWS IoT Greengrass 雲端服務，以擷取其可連線之核心裝置的相關資訊。然後，用戶端裝置可以嘗試連線到每個核心裝置，直到成功連線為止。

若要使用雲端探索，您必須執行下列動作：

- 將用戶端裝置與其可連線的核心裝置建立關聯。
- 指定用戶端裝置可以連線到每個核心裝置的 MQTT 代理程式端點。
- 將元件部署至支援用戶端裝置的核心裝置。

您也可以部署選用元件來執行下列動作：

- 在用戶端裝置、Greengrass 元件和 AWS IoT Core 雲端服務之間轉送訊息。
- 為您自動管理核心裝置 MQTT 代理程式端點。
- 管理本機用戶端裝置影子，並將影子與 AWS IoT Core 雲端服務同步。

您也必須檢閱和更新核心裝置 AWS IoT 的政策，以確認其具有連線用戶端裝置所需的許可。如需詳細資訊，請參閱[需求](#)。

設定雲端探索之後，您可以測試用戶端裝置與核心裝置之間的通訊。如需詳細資訊，請參閱[測試用戶端裝置通訊](#)。

主題

- [需求](#)
- [用戶端裝置支援的 Greengrass 元件](#)
- [設定雲端探索（主控台）](#)
- [設定雲端探索 \(AWS CLI\)](#)
- [關聯用戶端裝置](#)
- [離線時驗證用戶端](#)
- [管理核心裝置端點](#)
- [選擇 MQTT 代理程式](#)
- [使用 MQTT 代理程式將用戶端裝置連接至 AWS IoT Greengrass Core 裝置](#)
- [測試用戶端裝置通訊](#)
- [Greengrass 探索 RESTful API](#)

需求

若要將用戶端裝置連接至核心裝置，您必須具有下列項目：

- 核心裝置必須執行 [Greengrass 核 v2.2.0](#) 或更新版本。
- 在核心裝置運作 AWS 帳戶的 AWS 區域中，與 AWS IoT Greengrass 相關聯的 Greengrass 服務角色。如需詳細資訊，請參閱[設定 Greengrass 服務角色](#)。
- 核心裝置的 AWS IoT 政策必須允許下列許可：
 - `greengrass:PutCertificateAuthorities`

- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo` – (選用) 需要此許可才能使用 [IP 偵測器元件](#)，該元件會將核心裝置的網路連線資訊報告給 AWS IoT Greengrass 雲端服務。
- `iot:GetThingShadow`、`iot:UpdateThingShadow` 和 `iot:DeleteThingShadow` – (選用) 需要這些許可才能使用 [影子管理員元件](#) 來同步用戶端裝置影子 AWS IoT Core。此功能需要 [Greengrass 核](#) v2.6.0 或更新版本、[陰影管理員](#) v2.2.0 或更新版本，以及 [MQTT 橋接器](#) v2.2.0 或更新版本。

如需詳細資訊，請參閱[設定 AWS IoT 物件政策](#)。

Note

如果您在[安裝 AWS IoT Greengrass Core 軟體](#)時使用預設 AWS IoT 政策，則核心裝置具有允許存取所有 AWS IoT Greengrass 動作 AWS IoT 的政策 (`greengrass:*`)。

- AWS IoT 可以做為用戶端裝置連線的物件。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[建立 AWS IoT 資源](#)。
- 用戶端裝置必須使用用戶端 ID 連線。用戶端 ID 是物件名稱。不會接受其他用戶端 ID。
- 每個用戶端裝置的 AWS IoT 政策都必須允許 `greengrass:Discover` 許可。如需詳細資訊，請參閱[用戶端裝置的最低 AWS IoT 政策](#)。

主題

- [設定 Greengrass 服務角色](#)
- [設定 AWS IoT 物件政策](#)

設定 Greengrass 服務角色

Greengrass 服務角色是 AWS Identity and Access Management (IAM) 服務角色，授權代表您 AWS IoT Greengrass 從 AWS 服務存取資源。此角色可讓 AWS IoT Greengrass 驗證用戶端裝置的身分，並管理核心裝置連線資訊。

如果您先前尚未在此區域中設定 [Greengrass 服務角色](#)，則必須在此區域中 AWS IoT Greengrass 為建立 Greengrass 服務角色與 AWS 帳戶的關聯。

當您使用[AWS IoT Greengrass 主控台](#)中的設定核心裝置探索頁面時，會為您 AWS IoT Greengrass 設定 Greengrass 服務角色。否則，您可以使用[AWS IoT 主控台](#)或 AWS IoT Greengrass API 手動設定。

在本節中，您將檢查 Greengrass 服務角色是否已設定。如果未設定，您會在此 AWS 帳戶 區域中為建立新的 Greengrass AWS IoT Greengrass 服務角色，以與 建立關聯。

設定 Greengrass 服務角色（主控台）

1. 檢查 Greengrass 服務角色是否 AWS 帳戶 與此區域中 AWS IoT Greengrass 的 相關聯。請執行下列操作：
 - a. 導覽至 [AWS IoT 主控台](#)。
 - b. 在導覽窗格中，選擇設定。
 - c. 在 Greengrass 服務角色區段中，尋找目前的服務角色，以查看 Greengrass 服務角色是否相關聯。

如果您有相關聯的 Greengrass 服務角色，則需符合此要求才能使用 IP 偵測器元件。跳至 [設定 AWS IoT 物件政策](#)。

2. 如果 Greengrass 服務角色與此 AWS 帳戶 區域中 AWS IoT Greengrass 的 沒有關聯，請建立 Greengrass 服務角色並將其關聯。請執行下列操作：
 - a. 導覽至 [IAM 主控台](#)。
 - b. 選擇角色。
 - c. 選擇建立角色。
 - d. 在建立角色頁面上，執行下列動作：
 - i. 在信任的實體類型下，選擇 AWS 服務。
 - ii. 在使用案例、其他的使用案例 AWS 服務下，選擇 Greengrass，然後選取 Greengrass。此選項指定 將 新增 AWS IoT Greengrass 為可擔任此角色的信任實體。
 - iii. 選擇下一步。
 - iv. 在許可政策下，選取要AWSGreengrassResourceAccessRolePolicy連接至角色的。
 - v. 選擇下一步。
 - vi. 在角色名稱中，為您的角色輸入名稱，例如 **Greengrass_ServiceRole**。
 - vii. 選擇建立角色。

- e. 導覽至 [AWS IoT 主控台](#)。

- f. 在導覽窗格中，選擇設定。
- g. 在 Greengrass 服務角色區段中，選擇連接角色。
- h. 在更新 Greengrass 服務角色模態中，選取您建立的 IAM 角色，然後選擇連接角色。

設定 Greengrass 服務角色 (AWS CLI)

1. 檢查 Greengrass 服務角色是否 AWS 帳戶 與此區域中 AWS IoT Greengrass 的 相關聯。

```
aws greengrassv2 get-service-role-for-account
```

如果 Greengrass 服務角色相關聯，操作會傳回包含角色相關資訊的回應。

如果您有相關聯的 Greengrass 服務角色，則需符合此要求才能使用 IP 偵測器元件。跳至 [設定 AWS IoT 物件政策](#)。

2. 如果 Greengrass 服務角色與此 AWS 帳戶 區域中 AWS IoT Greengrass 的 沒有關聯，請建立 Greengrass 服務角色並將其關聯。請執行下列操作：
 - a. 建立具有信任政策的角色，AWS IoT Greengrass 允許 擔任該角色。此範例會建立名為 Greengrass_ServiceRole 的角色，但您可以使用不同的名稱。我們建議您也在信任政策中包含 aws:SourceArn 和 aws:SourceAccount 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

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. 從輸出中的角色中繼資料，複製角色 ARN。您使用 ARN 將角色與您的帳戶相關聯。

- c. 將 `AWSGreengrassResourceAccessRolePolicy` 政策連接到角色。

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

- d. 將 Greengrass 服務角色與 AWS IoT Greengrass 建立關聯 AWS 帳戶。將 `role-arn` 取代為服務角色的 ARN。

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

如果成功，操作會傳回下列回應。

```
{
  "associatedAt": "timestamp"
}
```

設定 AWS IoT 物件政策

核心裝置使用 X.509 裝置憑證來授權連線 AWS。您可以將 AWS IoT 政策連接至裝置憑證，以定義核心裝置的許可。如需詳細資訊，請參閱[AWS IoT 資料平面操作的 政策](#)及[支援用戶端裝置的最低 AWS IoT 政策](#)。

若要將用戶端裝置連線至核心裝置，核心裝置的 AWS IoT 政策必須允許下列許可：

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo` – (選用) 需要此許可才能使用 [IP 偵測器元件](#)，該元件會將核心裝置的網路連線資訊報告給 AWS IoT Greengrass 雲端服務。
- `iot:GetThingShadow`、`iot:UpdateThingShadow`和 `iot:DeleteThingShadow` – (選用) 使用[影子管理員元件](#)同步用戶端裝置影子時需要這些許可 AWS IoT Core。此功能需要 [Greengrass 核 v2.6.0](#) 或更新版本、[陰影管理員 v2.2.0](#) 或更新版本，以及 [MQTT 橋接器 v2.2.0](#) 或更新版本。

在本節中，您將檢閱核心裝置 AWS IoT 的政策，並新增任何缺少的必要許可。如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，您的核心裝置具有允許存取所有 AWS IoT Greengrass 動作

() AWS IoT 的政策 `greengrass:*`。在此情況下，只有在您計劃部署影子管理員元件以與裝置影子同步時，才必須更新 AWS IoT 政策 `AWS IoT Core`。否則，您可以略過本節。

設定 AWS IoT 物件政策（主控台）

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇核心裝置。
2. 在核心裝置頁面上，選擇要更新的核心裝置。
3. 在核心裝置詳細資訊頁面上，選擇核心裝置物件的連結。此連結會在 主控台中 AWS IoT 開啟物件詳細資訊頁面。
4. 在物件詳細資訊頁面上，選擇憑證。
5. 在憑證索引標籤中，選擇物件的作用中憑證。
6. 在憑證詳細資訊頁面上，選擇政策。
7. 在政策索引標籤中，選擇要檢閱和更新 AWS IoT 的政策。您可以將必要的許可新增至連接到核心裝置作用中憑證的任何政策。

Note

如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，則有兩個 AWS IoT 政策。如果 `GreengrassV2IoTThingPolicy` 存在，我們建議您選擇名為 的政策。您使用快速安裝程式建立的核心裝置預設會使用此政策名稱。如果您將許可新增至此政策，您也會將這些許可授予使用此政策的其他核心裝置。

8. 在政策概觀中，選擇編輯作用中版本。
9. 檢閱必要許可的政策，並新增任何缺少的必要許可。
 - `greengrass:PutCertificateAuthorities`
 - `greengrass:VerifyClientDeviceIdentity`
 - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
 - `greengrass:GetConnectivityInfo`
 - `greengrass:UpdateConnectivityInfo` –（選用）需要此許可才能使用 [IP 偵測器元件](#)，該元件會將核心裝置的網路連線資訊報告給 AWS IoT Greengrass 雲端服務。
 - `iot:GetThingShadow`、`iot:UpdateThingShadow` 和 `iot:DeleteThingShadow` –（選用）使用 [影子管理員元件](#) 同步用戶端裝置影子時需要這些許可 `AWS IoT Core`。此功能需要 [Greengrass 核 v2.6.0](#) 或更新版本、陰影管理員 `v2.2.0` 或更新版本，以及 [MQTT 橋接器 v2.2.0](#) 或更新版本。

10. (選用) 若要允許核心裝置與 同步陰影 AWS IoT Core，請將下列陳述式新增至政策。如果您計劃與用戶端裝置影子互動，但未與之同步 AWS IoT Core，請略過此步驟。將 *region* 和 *account-id* 取代為您使用的區域和您的 AWS 帳戶 號碼。
- 此範例陳述式允許存取所有物件的裝置影子。若要遵循最佳安全實務，您可以限制只能存取您連接至核心裝置的核心裝置和用戶端裝置。如需詳細資訊，請參閱[支援用戶端裝置的最低 AWS IoT 政策](#)。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

新增此陳述式後，政策文件看起來可能會類似下列範例。

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. 若要將新的政策版本設定為作用中版本，請在政策版本狀態下，選取將編輯的版本設定為此政策的作用中版本。
12. 選擇儲存為新版本。

設定 AWS IoT 物件政策 (AWS CLI)

1. 列出核心裝置 AWS IoT 物件的主體。物件主體可以是 X.509 裝置憑證或其他識別符。執行下列命令，並以核心裝置的名稱取代 *MyGreengrassCore*。

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

操作會傳回回應，列出核心裝置的物件主體。

```
{
  "principals": [
    "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
  ]
}
```

2. 識別核心裝置的作用中憑證。執行下列命令，並將 *certificateId* 取代為上一個步驟中每個憑證的 ID，直到您找到作用中憑證為止。憑證 ID 是憑證 ARN 結尾的十六進位字串。--query 引數指定 僅輸出憑證的狀態。

```
aws iot describe-certificate --certificate-id certificateId --query
'certificateDescription.status'
```

操作會以字串的形式傳回憑證狀態。例如，如果憑證處於作用中狀態，此操作會輸出 "ACTIVE"。

- 列出連接到憑證 AWS IoT 的政策。執行下列命令，並將憑證 ARN 取代為憑證的 ARN。

```
aws iot list-principal-policies --principal arn:aws:iot:us-west-2:123456789012:cert/certificateId
```

操作會傳回回應，列出連接到憑證 AWS IoT 的政策。

```
{
  "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"
    }
  ]
}
```

- 選擇要檢視和更新的政策。

Note

如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，則有兩個 AWS IoT 政策。如果 GreengrassV2IoTThingPolicy 存在，我們建議您選擇名為 `GreengrassV2IoTThingPolicy` 的政策。您使用快速安裝程式建立的核心裝置預設會使用此政策名稱。如果您將許可新增至此政策，您也會將這些許可授予使用此政策的其他核心裝置。

- 取得政策的文件。執行下列命令，並將 *GreengrassV2IoTThingPolicy* 取代為政策的名稱。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

操作會傳回回應，其中包含政策的文件和政策的其他資訊。政策文件是序列化為字串的 JSON 物件。

```
{
```

```

    "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. 使用線上轉換器或其他工具將政策文件字串轉換為 JSON 物件，然後將其儲存到名為 `iot-policy.json` 的檔案。

例如，如果您已安裝 [jq](#) 工具，您可以執行下列命令來取得政策文件、將其轉換為 JSON 物件，並將政策文件儲存為 JSON 物件。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
'policyDocument' | jq fromjson >> iot-policy.json
```

7. 檢閱必要許可的政策，並新增任何缺少的必要許可。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來開啟檔案。

```
nano iot-policy.json
```

- `greengrass:PutCertificateAuthorities`

- greengrass:VerifyClientDeviceIdentity
 - greengrass:VerifyClientDeviceIoTCertificateAssociation
 - greengrass:GetConnectivityInfo
 - greengrass:UpdateConnectivityInfo – (選用) 需要此許可才能使用 [IP 偵測器元件](#)，該元件會將核心裝置的網路連線資訊報告給 AWS IoT Greengrass 雲端服務。
 - iot:GetThingShadow、iot:UpdateThingShadow和 iot:DeleteThingShadow – (選用) 需要這些許可才能使用[影子管理員元件](#)來同步用戶端裝置影子 AWS IoT Core。此功能需要 [Greengrass 核](#) v2.6.0 或更新版本、陰影管理員 v2.2.0 或更新版本，以及 [MQTT 橋接器](#) v2.2.0 或更新版本。
8. 將變更儲存為政策的新版本。執行下列命令，並以政策的名稱取代 *GreengrassV2IoTThingPolicy*。

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-document file://iot-policy.json --set-as-default
```

如果成功，操作會傳回類似下列範例的回應。

```
{
  "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\\\": \\\"*\\\"\\
      }\\
    ]\\
  }\",
  "policyVersionId": "2",
  "isDefaultVersion": true
}
```

用戶端裝置支援的 Greengrass 元件

Important

核心裝置必須執行 [Greengrass 核 v2.2.0](#) 或更新版本，以支援用戶端裝置。

若要讓用戶端裝置能夠與核心裝置連線和通訊，您可以將下列 Greengrass 元件部署至核心裝置：

- [用戶端裝置身分驗證](#) (`aws.greengrass.clientdevices.Auth`)

部署用戶端裝置身分驗證元件，以驗證用戶端裝置並授權用戶端裝置動作。此元件可讓您的 AWS IoT 物件連線至核心裝置。

此元件需要一些組態才能使用它。您必須指定用戶端裝置的群組，以及每個群組獲授權執行的操作，例如透過 MQTT 連線和通訊。如需詳細資訊，請參閱[用戶端裝置驗證元件組態](#)。

- [MQTT 3.1.1 代理程式 \(Moquette\)](#) (`aws.greengrass.clientdevices.mqtt.Moquette`)

部署 Moquette MQTT 代理程式元件以執行輕量型 MQTT 代理程式。Moquette MQTT 代理程式符合 MQTT 3.1.1，並包含 QoS 0、QoS 1、QoS 2、保留訊息、最後遺囑訊息和持久性訂閱的本機支援。

您不需要設定此元件即可使用它。不過，您可以設定此元件操作 MQTT 代理程式的連接埠。根據預設，它會使用連接埠 8883。

- [MQTT 5 代理程式 \(EMQX\)](#) (`aws.greengrass.clientdevices.mqtt.EMQX`)

Note

若要使用 EMQX MQTT 5 代理程式，您必須使用 [Greengrass nucleus v2.6.0](#) 或更新版本，以及用戶端裝置驗證 v2.2.0 或更新版本。

部署 EMQX MQTT 代理程式元件，以在用戶端裝置和核心裝置之間使用 MQTT 5.0 功能進行通訊。EMQX MQTT 代理程式符合 MQTT 5.0，包括對工作階段和訊息過期間隔、使用者屬性、共用訂閱、主題別名等的支援。

您不需要設定此元件即可使用它。不過，您可以設定此元件操作 MQTT 代理程式的連接埠。根據預設，它會使用連接埠 8883。

- [MQTT 橋接器](#) (`aws.greengrass.clientdevices.mqtt.Bridge`)

(選用) 部署 MQTT 橋接器元件，以在用戶端裝置 (本機 MQTT)、本機發佈/訂閱和 AWS IoT Core MQTT 之間轉送訊息。設定此元件以同步用戶端裝置與 Greengrass 元件中的用戶端裝置，AWS IoT Core 並與之互動。

此元件需要使用組態。您必須指定此元件轉送訊息的主題映射。如需詳細資訊，請參閱 [MQTT 橋接器元件組態](#)。

- [IP 偵測器](#) (`aws.greengrass.clientdevices.IPDetector`)

(選用) 部署 IP 偵測器元件，以自動向 AWS IoT Greengrass 雲端服務報告核心裝置的 MQTT 代理程式端點。如果您有複雜的網路設定，例如路由器將 MQTT 代理程式連接埠轉送至核心裝置的網路設定，則無法使用此元件。

您不需要設定此元件即可使用它。

- [Shadow 管理員](#) (`aws.greengrass.ShadowManager`)

 Note

若要管理用戶端裝置影子，您必須使用 [Greengrass 核 v2.6.0](#) 或更新版本、影子管理員 v2.2.0 或更新版本，以及 [MQTT 橋接器 v2.2.0](#) 或更新版本。

(選用) 部署影子管理員元件以管理核心裝置上的用戶端裝置影子。Greengrass 元件可以取得、更新和刪除用戶端裝置影子，以與用戶端裝置互動。您也可以設定影子管理員元件，將用戶端裝置影子與 AWS IoT Core 雲端服務同步。

若要將此元件與用戶端裝置影子搭配使用，您必須設定 MQTT 橋接器元件，在用戶端裝置與使用本機發佈/訂閱的影子管理員之間轉送訊息。否則，此元件不需要組態即可使用，但需要組態才能同步裝置陰影。

 Note

我們建議您僅部署一個 MQTT 代理程式元件。[MQTT 橋接器](#)和 [IP 偵測器](#)元件一次只能使用一個 MQTT 代理程式元件。如果您部署多個 MQTT 代理程式元件，則必須將其設定為使用不同的連接埠。

設定雲端探索（主控台）

您可以使用 AWS IoT Greengrass 主控台來關聯用戶端裝置、管理核心裝置端點，以及部署元件以啟用用戶端裝置支援。如需詳細資訊，請參閱[步驟 2：啟用用戶端裝置支援](#)。

設定雲端探索 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 來關聯用戶端裝置、管理核心裝置端點，以及部署元件以啟用用戶端裝置支援。如需詳細資訊，請參閱下列內容：

- [管理用戶端裝置關聯 \(AWS CLI\)](#)
- [管理核心裝置端點](#)
- [AWS提供的用戶端裝置元件](#)
- [建立部署](#)

關聯用戶端裝置

若要使用雲端探索，請將用戶端裝置與核心裝置建立關聯，以便他們可以探索核心裝置。然後，他們可以使用 [Greengrass 探索 API](#) 來擷取其相關聯核心裝置的連線資訊和憑證。

同樣地，取消用戶端裝置與核心裝置的關聯，使其無法探索核心裝置。

主題

- [管理用戶端裝置關聯（主控台）](#)
- [管理用戶端裝置關聯 \(AWS CLI\)](#)
- [管理用戶端裝置關聯 \(API\)](#)

管理用戶端裝置關聯（主控台）

您可以使用 AWS IoT Greengrass 主控台來檢視、新增和刪除用戶端裝置關聯。

檢視核心裝置的用戶端裝置關聯（主控台）

1. 導覽至 [AWS IoT Greengrass 主控台](#)。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資訊頁面上，選擇用戶端裝置索引標籤。

5. 在關聯的用戶端裝置區段中，您可以查看哪些用戶端裝置 (AWS IoT 物件) 與核心裝置相關聯。

將用戶端裝置與核心裝置建立關聯 (主控台)

1. 導覽至 [AWS IoT Greengrass 主控台](#)。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資訊頁面上，選擇用戶端裝置索引標籤。
5. 在關聯的用戶端裝置區段中，選擇關聯用戶端裝置。
6. 在將用戶端裝置與核心裝置模式建立關聯中，為每個要建立關聯的用戶端裝置執行下列動作：
 - a. 輸入要做為用戶端裝置建立關聯的 AWS IoT 物件名稱。
 - b. 選擇新增。
7. 選擇關聯。

您相關聯的用戶端裝置現在可以使用 Greengrass 探索 API 來探索此核心裝置。

取消用戶端裝置與核心裝置的關聯 (主控台)

1. 導覽至 [AWS IoT Greengrass 主控台](#)。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資訊頁面上，選擇用戶端裝置索引標籤。
5. 在關聯的用戶端裝置區段中，選取要取消關聯的每個用戶端裝置。
6. 選擇取消關聯。
7. 在確認模式，選擇取消關聯。

您取消關聯的用戶端裝置無法再使用 Greengrass 探索 API 來探索此核心裝置。

管理用戶端裝置關聯 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 來管理核心裝置的用戶端裝置關聯。

檢視核心裝置的用戶端裝置關聯 (AWS CLI)

- 使用下列命令：[list-client-devices-associated-with-core-device](#)。

將用戶端裝置與核心裝置建立關聯 (AWS CLI)

- 使用下列命令：[batch-associate-client-device-with-core-device](#)。

取消用戶端裝置與核心裝置的關聯 (AWS CLI)

- 使用下列命令：[batch-disassociate-client-device-from-core-device](#)。

管理用戶端裝置關聯 (API)

您可以使用 AWS API 來管理核心裝置的用戶端裝置關聯。

檢視核心裝置 (AWS API) 的用戶端裝置關聯

- 使用下列操作：[ListClientDevicesAssociatedWithCoreDevice](#)。

將用戶端裝置與核心裝置 (AWS API) 建立關聯

- 使用下列操作：[BatchAssociateClientDeviceWithCoreDevice](#)。

取消用戶端裝置與核心裝置 (AWS API) 的關聯

- 使用下列操作：[BatchDisassociateClientDeviceFromCoreDevice](#)。

離線時驗證用戶端

透過離線身分驗證，您可以設定您的 AWS IoT Greengrass Core 裝置，讓用戶端裝置可以連線至核心裝置，即使核心裝置未連線至雲端也一樣。當您使用離線身分驗證時，您的 Greengrass 裝置可以繼續在部分離線的環境中運作。

若要對具有雲端連線的用戶端裝置使用離線身分驗證，您需要下列項目：

- 部署 [用戶端裝置身分驗證](#) 元件的核心 AWS IoT Greengrass 裝置。您必須使用 2.3.0 版或更新版本進行離線身分驗證。
- 在用戶端裝置的初始連線期間，核心裝置的雲端連線。

儲存用戶端登入資料

當用戶端裝置第一次連線到核心裝置時，核心裝置會呼叫 AWS IoT Greengrass 服務。呼叫時，Greengrass 會將用戶端裝置的註冊驗證為 AWS IoT 物件。它也會驗證裝置是否具有有效的憑證。然後，核心裝置會在本機存放此資訊。

下次裝置連線時，Greengrass 核心裝置會嘗試使用 AWS IoT Greengrass 服務驗證用戶端裝置。如果無法連線 AWS IoT Greengrass，核心裝置會使用其本機儲存的裝置資訊來驗證用戶端裝置。

您可以設定 Greengrass 核心裝置存放登入資料的時間長度。您可以在[用戶端裝置驗證元件組態](#)中設定 `clientDeviceTrustDurationMinutes` 組態選項，將逾時從一分鐘設定為 2，147，483，647 分鐘。預設值為一分鐘，可有效關閉離線身分驗證。當您設定此逾時時，我們建議您考慮安全需求。您也應該考慮在與雲端中斷連線時，預期核心裝置執行的時間長度。

核心裝置會更新其登入資料儲存三次：

1. 當裝置第一次連接到核心裝置時。
2. 如果核心裝置連接到雲端，當用戶端裝置重新連接到核心裝置時。
3. 如果核心裝置已連線至雲端，請每天重新整理一次整個登入資料存放區。

當 Greengrass 核心裝置重新整理其登入資料存放區時，會使用 [ListClientDevicesAssociatedWithCoreDevice](#) 操作。Greengrass 只會重新整理此操作傳回的裝置。若要將用戶端裝置與核心裝置建立關聯，請參閱 [關聯用戶端裝置](#)。

若要使用 `ListClientDevicesAssociatedWithCoreDevice` 操作，您必須將操作的許可新增至與執行相關聯的 AWS Identity and Access Management (IAM) AWS 帳戶 角色 AWS IoT Greengrass。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

管理核心裝置端點

當您使用雲端探索時，您可以將核心裝置的 MQTT 代理程式端點存放在雲端服務中 AWS IoT Greengrass。用戶端裝置會連線至 AWS IoT Greengrass，以擷取這些端點及其相關聯核心裝置的其他資訊。

對於每個核心裝置，您可以自動或手動管理端點。

- 使用 IP 偵測器自動管理端點

如果您有非複雜網路設定，例如用戶端裝置與核心裝置位於相同網路上，您可以部署 [IP 偵測器元件](#)來自動管理核心裝置端點。例如，如果核心裝置位於將 MQTT 代理程式連接埠轉送至核心裝置的路由器後方，則無法使用 IP 偵測器元件。

如果您部署到物件群組，IP 偵測器元件也很有用，因為它會管理物件群組中所有核心裝置的端點。如需詳細資訊，請參閱[使用 IP 偵測器自動管理端點](#)。

- 手動管理端點

如果您無法使用 IP 偵測器元件，則必須手動管理核心裝置端點。您可以使用 主控台或 API 更新這些端點。如需詳細資訊，請參閱[手動管理端點](#)。

主題

- [使用 IP 偵測器自動管理端點](#)
- [手動管理端點](#)

使用 IP 偵測器自動管理端點

如果您有簡單的網路設定，例如與核心裝置位於相同網路上的用戶端裝置，您可以部署 [IP 偵測器元件](#)來執行下列動作：

- 監控 Greengrass 核心裝置的本機網路連線資訊。此資訊包括核心裝置的網路端點，以及 MQTT 中介裝置操作所在的連接埠。
- 向 AWS IoT Greengrass 雲端服務報告核心裝置的連線資訊。

IP 偵測器元件會覆寫您手動設定的端點。

Important

核心裝置的 AWS IoT 政策必須允許 使用 IP 偵測器元件的 `greengrass:UpdateConnectivityInfo` 許可。如需詳細資訊，請參閱[AWS IoT 資料平面操作的 政策及設定 AWS IoT 物件政策](#)。

您可以執行下列任一動作來部署 IP 偵測器元件：

- 使用 主控台 中的設定探索頁面。如需詳細資訊，請參閱[設定雲端探索（主控台）](#)。

- 建立和修訂部署以包含 IP 偵測器。您可以使用 主控台 AWS CLI、或 AWS API 來管理部署。如需詳細資訊，請參閱[建立部署](#)。

部署 IP 偵測器元件（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
2. 在元件頁面上，選擇公有元件索引標籤，然後選擇 `aws.greengrass.clientdevices.IPDetector`。
3. 在 `aws.greengrass.clientdevices.IPDetector` 頁面中，選擇部署。
4. 從新增至部署中，選擇要修改的現有部署，或選擇建立新的部署，然後選擇下一步。
5. 如果您選擇建立新的部署，請選擇部署的目標核心裝置或物件群組。在指定目標頁面的部署目標下，選擇核心裝置或物件群組，然後選擇下一步。
6. 在選取元件頁面上，確認已選取 `aws.greengrass.clientdevices.IPDetector` 元件，選擇下一步。
7. 在設定元件頁面上，選取 `aws.greengrass.clientdevices.IPDetector`，然後執行下列動作：
 - a. 選擇設定元件。
 - b. 在設定 `aws.greengrass.clientdevices.IPDetector` 模態、組態更新下，在要合併的組態中，您可以輸入組態更新來設定 IP 偵測器元件。您可以指定下列任何組態選項：
 - `defaultPort` –（選用）此元件偵測到 IP 地址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。
 - `includeIPv4LoopbackAddrs` –（選用）您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。
 - `includeIPv4LinkLocalAddrs` –（選用）您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。
 - `includeIPv6LoopbackAddrs` –（選用）您可以啟用此選項來偵測和報告 IPv6 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
 - `includeIPv6LinkLocalAddrs` –（選用）您可以啟用此選項來偵測和報告 IPv6 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs`

設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。

- `includeIPv4Addrs` – (選用) 預設值設定為 `true`。您可以啟用此選項來發佈核心裝置上找到的 IPv4 地址。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6Addrs` – (選用) 您可以啟用此選項來發佈核心裝置上找到的 IPv6 地址。`includeIPv4Addrs` 設定為 `false` 以使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。

組態更新看起來可能會類似下列範例。

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false
}
```

- c. 選擇確認以關閉模態，然後選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
 9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署可能需要一分鐘的時間才能完成。

部署 IP 偵測器元件 (AWS CLI)

若要部署 IP 偵測器元件，請建立包含在 `components` 物件 `aws.greengrass.clientdevices.IPDetector` 中的部署文件，並指定元件的組態更新。遵循中的指示 [建立部署](#) 建立新的部署或修改現有的部署。

您可以在建立部署文件時指定下列任一選項來設定 IP 偵測器元件：

- `defaultPort` – (選用) 此元件偵測到 IP 地址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。
- `includeIPv4LoopbackAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。
- `includeIPv4LinkLocalAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。

- `includeIPv6LoopbackAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv6 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6LinkLocalAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv6 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv4Addrs` – (選用) 預設值設定為 `true`。您可以啟用此選項來發佈核心裝置上找到的 IPv4 地址。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6Addrs` – (選用) 您可以啟用此選項來發佈核心裝置上找到的 IPv6 地址。`includeIPv4Addrs` 設定為 `false` 以使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。

下列範例部分部署文件指定 將連接埠 8883 報告為 MQTT 代理程式連接埠。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\"}"
      }
    }
  }
}
```

手動管理端點

您可以手動管理核心裝置的 MQTT 代理程式端點。

每個 MQTT 代理程式端點都有以下資訊：

端點 (HostAddress)

用戶端裝置可以連線到核心裝置上 MQTT 代理程式的 IP 地址或 DNS 地址。

連接埠 (PortNumber)

MQTT 中介裝置在核心裝置上操作的連接埠。

您可以在預設使用連接埠 8883 的 [Moquette MQTT 代理程式元件](#) 上設定此連接埠。

中繼資料 (Metadata)

要提供給連接到此端點之用戶端裝置的其他中繼資料。

主題

- [管理端點（主控台）](#)
- [管理端點 \(AWS CLI\)](#)
- [管理端點 \(API\)](#)

管理端點（主控台）

您可以使用 AWS IoT Greengrass 主控台來檢視、更新和移除核心裝置的端點。

管理核心裝置的端點（主控台）

1. 導覽至 [AWS IoT Greengrass 主控台](#)。
2. 選擇核心裝置。
3. 選擇要管理的核心裝置。
4. 在核心裝置的詳細資訊頁面上，選擇用戶端裝置索引標籤。
5. 在 MQTT 代理程式端點區段中，您可以查看核心裝置的 MQTT 代理程式端點。選擇管理端點。
6. 在管理端點模態中，新增或移除核心裝置的 MQTT 代理程式端點。
7. 選擇更新。

管理端點 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 來管理核心裝置的端點。

 Note

由於 中的用戶端裝置支援 AWS IoT Greengrass V2 與 回溯相容 AWS IoT Greengrass V1，您可以使用 AWS IoT Greengrass V2 或 AWS IoT Greengrass V1 API 操作來管理核心裝置端點。

取得核心裝置的端點 (AWS CLI)

- 使用下列其中一個命令：
 - [greengrassv2 : get-connectivity-info](#)
 - [greengrass : get-connectivity-info](#)

更新核心裝置的端點 (AWS CLI)

- 使用下列其中一個命令：
 - [greengrassv2 : update-connectivity-info](#)
 - [greengrass : update-connectivity-info](#)

管理端點 (API)

您可以使用 AWS API 來管理核心裝置的端點。

 Note

由於 中的用戶端裝置支援 AWS IoT Greengrass V2 與 回溯相容 AWS IoT Greengrass V1，您可以使用 AWS IoT Greengrass V2 或 AWS IoT Greengrass V1 API 操作來管理核心裝置端點。

取得核心裝置 (AWS API) 的端點

- 使用下列任一操作：
 - [V2 : GetConnectivityInfo](#)
 - [V1 : GetConnectivityInfo](#)

更新核心裝置 (AWS API) 的端點

- 使用下列任一操作：
 - [V2 : UpdateConnectivityInfo](#)
 - [V1 : UpdateConnectivityInfo](#)

選擇 MQTT 代理程式

AWS IoT Greengrass 提供選項，讓您選擇要在核心裝置上執行的本機 MQTT 代理程式。用戶端裝置會連線至核心裝置上執行的 MQTT 代理程式，因此請選擇與您要連線的用戶端裝置相容的 MQTT 代理程式。

Note

我們建議您僅部署一個 MQTT 代理程式元件。[MQTT 橋接器](#)和 [IP 偵測器](#)元件一次只能使用一個 MQTT 代理程式元件。如果您部署多個 MQTT 代理程式元件，則必須將其設定為使用不同的連接埠。

您可以從下列 MQTT 代理程式中選擇：

- [MQTT 3.1.1 代理程式 \(Moquette\)](#) – `aws.greengrass.clientdevices.mqtt.Moquette`

針對符合 MQTT 3.1.1 標準的輕量型 MQTT 代理程式選擇此選項。AWS IoT Core MQTT 代理程式和 AWS IoT Device SDK 也符合 MQTT 3.1.1 標準，因此您可以使用這些功能來建立應用程式，在您的裝置和中使用 MQTT 3.1.1。AWS 雲端

- [MQTT 5 代理程式 \(EMQX\)](#) – `aws.greengrass.clientdevices.mqtt.EMQX`

選擇此選項以使用核心裝置和用戶端裝置之間的 MQTT 5 功能進行通訊。此元件使用的資源比 Moquette MQTT 3.1.1 代理程式多，在 Linux 核心裝置上需要 Docker。

MQTT 5 與 MQTT 3.1.1 向後相容，因此您可以將使用 MQTT 3.1.1 的用戶端裝置連接到此代理程式。如果您執行 Moquette MQTT 3.1.1 代理程式，則可以將其取代為 EMQX MQTT 5 代理程式，而且用戶端裝置可以繼續如常連線和操作。

- 實作自訂代理程式

選擇此選項可建立自訂本機代理程式元件，以與用戶端裝置通訊。您可以建立使用 MQTT 以外通訊協定的自訂本機代理程式。AWS IoT Greengrass 提供元件 SDK，可用來驗證和授權用戶端裝置。如需詳細資訊，請參閱 [使用 AWS IoT Device SDK 與 Greengrass 核、其他元件和 通訊 AWS IoT Core](#) 和 [驗證和授權用戶端裝置](#)。

使用 MQTT 代理程式將用戶端裝置連接至 AWS IoT Greengrass Core 裝置

當您在 AWS IoT Greengrass Core 裝置上使用 MQTT 代理程式時，裝置會使用裝置特有的核心裝置憑證授權機構 (CA)，向代理程式發出憑證，以便與用戶端進行交互 TLS 連線。

AWS IoT Greengrass 會自動產生核心裝置 CA，您也可以提供自己的 CA。連接[用戶端裝置身分驗證](#)元件 AWS IoT Greengrass 時，核心裝置 CA 會向註冊。自動產生的核心裝置 CA 為持久性，只要已設定用戶端裝置驗證元件，裝置就會繼續使用相同的 CA。

當 MQTT 代理程式啟動時，它會請求憑證。用戶端裝置身分驗證元件會使用核心裝置 CA 發行 X.509 憑證。代理程式啟動、憑證過期或 IP 地址等連線資訊變更時，會輪換憑證。如需詳細資訊，請參閱[本機 MQTT 代理程式上的憑證輪換](#)。

若要將用戶端連線至 MQTT 代理程式，您需要下列項目：

- 用戶端裝置必須具有 AWS IoT Greengrass Core 裝置 CA。您可以透過雲端探索或手動提供 CA 來取得此 CA。如需詳細資訊，請參閱[使用您自己的憑證授權單位](#)。
- 核心裝置的完整網域名稱 (FQDN) 或 IP 地址必須存在於核心裝置 CA 發行的代理程式憑證中。您可以使用 [IP 偵測器](#) 元件或手動設定 IP 地址來確保這一點。如需詳細資訊，請參閱[管理核心裝置端點](#)。
- 用戶端裝置身分驗證元件必須授予用戶端裝置連線至 Greengrass 核心裝置的許可。如需詳細資訊，請參閱[用戶端裝置身分驗證](#)。

使用您自己的憑證授權單位

如果您的用戶端裝置無法存取雲端來探索您的核心裝置，您可以提供核心裝置憑證授權單位 (CA)。您的 Greengrass 核心裝置使用核心裝置 CA 為您的 MQTT 代理程式發行憑證。設定核心裝置並以其 CA 佈建用戶端裝置後，您的用戶端裝置就可以連線至端點，並使用核心裝置 CA（已提供 CA 或自動產生）驗證 TLS 交換。

若要設定[用戶端裝置身分驗證](#)元件以使用您的核心裝置 CA，請在部署元件時設定 `certificateAuthority` 組態參數。您必須在組態期間提供下列詳細資訊：

- 核心裝置 CA 憑證的位置。
- 核心裝置 CA 憑證的私有金鑰。
- (選用) 如果核心裝置 CA 是中繼 CA，則為根憑證的憑證鏈結。

如果您提供核心裝置 CA，會將 CA AWS IoT Greengrass 註冊至雲端。

您可以將憑證存放在硬體安全模組或檔案系統上。下列範例顯示使用 HSM/TPM 存放的中繼 CA `certificateAuthority` 組態。請注意，憑證鏈只能存放在磁碟上。

```
"certificateAuthority": {  
  "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",  
  "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"  
  "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",  
}
```

在此範例中，`certificateAuthority` 組態參數會將用戶端裝置身分驗證元件設定為使用檔案系統的中繼 CA：

```
"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",  
}
```

若要將裝置連接到您的 AWS IoT Greengrass Core 裝置，請執行下列動作：

1. 使用組織的根 CA 為 Greengrass 核心裝置建立中繼憑證授權機構 (CA)。我們建議您使用中繼 CA 做為安全最佳實務。
2. 將中繼 CA 憑證、私有金鑰和憑證鏈提供給 Greengrass 核心裝置的根 CA。如需詳細資訊，請參閱[用戶端裝置身分驗證](#)。中繼 CA 會成為 Greengrass 核心裝置的核心裝置 CA，而裝置會向該 CA 註冊 AWS IoT Greengrass。
3. 將用戶端裝置註冊為 AWS IoT 物件。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[建立物件](#)。將私有金鑰、公有金鑰、裝置憑證和根 CA 憑證新增至您的用戶端裝置。如何新增資訊取決於您的裝置和軟體。

設定裝置後，您可以使用憑證和公有金鑰鏈連線到 Greengrass 核心裝置。您的軟體負責尋找核心裝置端點。您可以手動設定核心裝置的端點。如需詳細資訊，請參閱[手動管理端點](#)。

測試用戶端裝置通訊

用戶端裝置可以使用 AWS IoT Device SDK 來探索、連線核心裝置，以及與核心裝置通訊。您可以在中使用 Greengrass 探索用戶端 AWS IoT Device SDK 來使用 [Greengrass 探索 API](#)，這會傳回用戶端裝置可連線之核心裝置的相關資訊。API 回應包含要連線的 MQTT 代理程式端點，以及用來驗證每個核心裝置的身分的憑證。然後，用戶端裝置可以嘗試每個端點，直到成功連接到核心裝置為止。

用戶端裝置只能探索與之建立關聯的核心裝置。在測試用戶端裝置與核心裝置之間的通訊之前，您必須將用戶端裝置與核心裝置建立關聯。如需詳細資訊，請參閱[關聯用戶端裝置](#)。

Greengrass 探索 API 會傳回您指定的核心裝置 MQTT 代理程式端點。您可以使用 [IP 偵測器元件](#)來為您管理這些端點，也可以為每個核心裝置手動管理它們。如需詳細資訊，請參閱[管理核心裝置端點](#)。

Note

若要使用 Greengrass 探索 API，用戶端裝置必須具有 `greengrass:Discover` 許可。如需詳細資訊，請參閱[用戶端裝置的最低 AWS IoT 政策](#)。

AWS IoT Device SDK 提供多種程式設計語言。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南[AWS IoT](#)》中的 [裝置 SDKs](#)。

主題

- [測試通訊 \(Python\)](#)
- [測試通訊 \(C++\)](#)
- [測試通訊 \(JavaScript\)](#)
- [測試通訊 \(Java\)](#)

測試通訊 (Python)

在本節中，您會在適用於 [AWS IoT Device SDK Python 的 v2](#) 中使用 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

Important

若要使用 AWS IoT Device SDK v2 for Python，裝置必須執行 Python 3.6 或更新版本。

測試通訊 (AWS IoT Device SDK v2 for Python)

1. 下載並安裝適用於 [AWS IoT Device SDK Python 的 v2](https://github.com/aws/aws-iot-device-sdk-python-v2) 至 AWS IoT 物件，以做為用戶端裝置進行連線。

在用戶端裝置上，執行下列動作：

- a. 複製 AWS IoT Device SDK v2 for Python 儲存庫以進行下載。

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

- b. 安裝 AWS IoT Device SDK v2 for Python。

```
python3 -m pip install --user ./aws-iot-device-sdk-python-v2
```

2. 變更為 AWS IoT Device SDK v2 for Python 中的範例資料夾。

```
cd aws-iot-device-sdk-python-v2/samples/greengrass
```

3. 執行範例 Greengrass 探索應用程式。此應用程式需要指定用戶端裝置物件名稱的引數、要使用的 MQTT 主題和訊息，以及驗證和保護連線的憑證。下列範例會將 Hello World 訊息傳送至 `clients/MyClientDevice1/hello/world` 主題。

- 將 `MyClientDevice1` 取代為用戶端裝置的物件名稱。
- 將 `~/certs/AmazonRootCA1.pem` 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
- 將 `~/certs/device.pem.crt` 取代為用戶端裝置上裝置憑證的路徑。
- 將 `~/certs/private.pem.key` 取代為用戶端裝置上私有金鑰檔案的路徑。
- 將 `us-east-1` 取代為用戶端裝置和核心裝置操作所在的 AWS 區域。

```
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
```


探索範例應用程式會傳送訊息 10 次並中斷連線。它也會訂閱發佈訊息的相同主題。如果輸出指出應用程式收到有關主題的 MQTT 訊息，用戶端裝置可以成功與核心裝置通訊。

```
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}'
```

如果應用程式輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

您也可以在核心裝置上檢視 Greengrass 日誌，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

測試通訊 (C++)

在本節中，您會在適用於 [AWS IoT Device SDK C++ 的 v2](#) 中使用 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

若要建置適用於 C++ 的 AWS IoT Device SDK v2，裝置必須具有下列工具：

- C++ 11 或更新版本
- CMake 3.1 或更新版本
- 下列其中一個編譯器：
 - GCC 4.8 或更新版本
 - Clang 3.9 或更新版本
 - MSVC 2015 或更新版本

測試通訊（適用於 C++ 的 AWS IoT Device SDK v2）

1. 下載並建置適用於 [AWS IoT Device SDK C++ 的 v2](#) 至 AWS IoT 物件，以做為用戶端裝置進行連線。

在用戶端裝置上，執行下列動作：

- a. 建立適用於 C++ 工作區的 AWS IoT Device SDK v2 資料夾，然後變更為該資料夾。

```
cd
mkdir iot-device-sdk-cpp
cd iot-device-sdk-cpp
```

- b. 複製適用於 C++ 儲存庫的 AWS IoT Device SDK v2 以進行下載。--recursive 旗標指定下載子模組。

```
git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
```

- c. 為 AWS IoT Device SDK v2 建立 C++ 組建輸出的資料夾，然後變更為該資料夾。

```
mkdir aws-iot-device-sdk-cpp-v2-build
cd aws-iot-device-sdk-cpp-v2-build
```

- d. 建置適用於 C++ 的 AWS IoT Device SDK v2。

```
cmake -DCMAKE_INSTALL_PREFIX="/iot-device-sdk-cpp" -  
DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2  
cmake --build . --target install
```

2. 在適用於 C++ 的 AWS IoT Device SDK v2 中建置 Greengrass 探索範例應用程式。請執行下列操作：

- a. 變更為適用於 C++ 的 AWS IoT Device SDK v2 中的 Greengrass 探索範例資料夾。

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

- b. 建立 Greengrass 探索範例建置輸出的資料夾，然後變更為該資料夾。

```
mkdir build  
cd build
```

- c. 建置 Greengrass 探索範例應用程式。

```
cmake -DCMAKE_PREFIX_PATH="/iot-device-sdk-cpp" -  
DCMAKE_BUILD_TYPE="Release" ..  
cmake --build . --config "Release"
```

3. 執行範例 Greengrass 探索應用程式。此應用程式預期引數會指定用戶端裝置物件名稱、要使用的 MQTT 主題，以及驗證和保護連線的憑證。下列範例會訂閱 `clients/MyClientDevice1/hello/world` 主題，並將您在命令列中輸入的訊息發佈至相同的主題。

- 將 `MyClientDevice1` 取代為用戶端裝置的物件名稱。
- 將 `~/certs/AmazonRootCA1.pem` 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
- 將 `~/certs/device.pem.crt` 取代為用戶端裝置上裝置憑證的路徑。
- 將 `~/certs/private.pem.key` 取代為用戶端裝置上私有金鑰檔案的路徑。
- 將 `us-east-1` 取代為用戶端裝置和核心裝置操作所在的 AWS 區域。

```
./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
```

探索範例應用程式會訂閱主題，並提示您輸入要發佈的訊息。

```
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.
```

如果應用程式輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

4. 輸入訊息，例如 **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!
```

如果輸出指出應用程式收到主題的 MQTT 訊息，用戶端裝置可以成功與核心裝置通訊。

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

您也可以核心裝置上檢視 Greengrass 日誌，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

測試通訊 (JavaScript)

在本節中，您會在適用於 [AWS IoT Device SDK JavaScript 的 v2](#) 中使用 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

Important

若要使用 AWS IoT Device SDK v2 for JavaScript，裝置必須執行 Node v10.0 或更新版本。

測試通訊（適用於 JavaScript 的 AWS IoT Device SDK v2）

1. 下載並安裝適用於 [AWS IoT Device SDK JavaScript 的 v2](#) 至 AWS IoT 物件，以做為用戶端裝置進行連線。

在用戶端裝置上，執行下列動作：

- a. 複製 AWS IoT Device SDK v2 for JavaScript 儲存庫以進行下載。

```
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

- b. 安裝適用於 JavaScript 的 AWS IoT Device SDK v2。

```
cd aws-iot-device-sdk-js-v2
npm install
```

2. 變更為適用於 JavaScript 的 AWS IoT Device SDK v2 中的 Greengrass 探索範例資料夾。

```
cd samples/node/greengrass/basic_discovery
```

3. 安裝 Greengrass 探索範例應用程式。

```
npm install
```

4. 執行範例 Greengrass 探索應用程式。此應用程式需要指定用戶端裝置物件名稱的引數、要使用的 MQTT 主題和訊息，以及驗證和保護連線的憑證。下列範例會將 Hello World 訊息傳送至 `clients/MyClientDevice1/hello/world` 主題。

- 將 `MyClientDevice1` 取代為用戶端裝置的物件名稱。
- 將 `~/certs/AmazonRootCA1.pem` 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
- 將 `~/certs/device.pem.crt` 取代為用戶端裝置上裝置憑證的路徑。
- 將 `~/certs/private.pem.key` 取代為用戶端裝置上私有金鑰檔案的路徑。
- 將 `us-east-1` 取代為您的用戶端裝置和核心裝置操作 AWS 的區域。

```
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
```

探索範例應用程式會傳送訊息 10 次並中斷連線。它也會訂閱發佈訊息的相同主題。如果輸出指出應用程式收到有關主題的 MQTT 訊息，用戶端裝置可以成功與核心裝置通訊。

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":""}]}],"certificate_chain":["-----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!
```

如果應用程式輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

您也可以在核心裝置上檢視 Greengrass 日誌，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

測試通訊 (Java)

在本節中，您會在適用於 [AWS IoT Device SDK Java 的 v2](#) 中使用 Greengrass 探索範例來測試用戶端裝置與核心裝置之間的通訊。

Important

若要建置適用於 Java 的 AWS IoT Device SDK v2，裝置必須具有下列工具：

- Java 8 或更新版本，並 JAVA_HOME 指向 Java 資料夾。
- Apache Maven

測試通訊 (AWS IoT Device SDK v2 for Java)

1. 下載並建置適用於 [AWS IoT Device SDK Java 的 v2](#) 至 AWS IoT 物件，以做為用戶端裝置進行連線。

在用戶端裝置上，執行下列動作：

- a. 複製 AWS IoT Device SDK v2 for Java 儲存庫以進行下載。

```
git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
```

- b. 變更為 AWS IoT Device SDK v2 for Java 資料夾。

c. 建置適用於 Java 的 AWS IoT Device SDK v2。

```
cd aws-iot-device-sdk-java-v2
mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
mvn clean install
```

2. 執行範例 Greengrass 探索應用程式。此應用程式需要指定用戶端裝置物件名稱的引數、要使用的 MQTT 主題，以及驗證和保護連線的憑證。下列範例會訂閱clients/*MyClientDevice1*/hello/world主題，並將您在命令列中輸入的訊息發佈至相同的主題。

- 以用戶端裝置的物件名稱取代 *MyClientDevice1* 的兩個執行個體。
- 將 *\$HOME/certs/AmazonRootCA1.pem* 取代為用戶端裝置上 Amazon 根 CA 憑證的路徑。
- 將 *\$HOME/certs/device.pem.crt* 取代為用戶端裝置上裝置憑證的路徑。
- 將 *\$HOME/certs/private.pem.key* 取代為用戶端裝置上的私有金鑰檔案路徑。
- 將 *us-east-1* 取代為您的用戶端裝置和核心裝置操作 AWS 區域 所在的。

```
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"
```

探索範例應用程式會訂閱主題，並提示您輸入要發佈的訊息。

```
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:
```

如果應用程式輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

3. 輸入訊息，例如 **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!
```

如果輸出指出應用程式收到主題的 MQTT 訊息，用戶端裝置可以成功與核心裝置通訊。

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

您也可以在核心裝置上檢視 Greengrass 日誌，以驗證用戶端裝置是否成功連線並傳送訊息。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

Greengrass 探索 RESTful API

AWS IoT Greengrass 提供用戶端裝置可用來識別 Greengrass 核心裝置連線的 Discover API 操作。用戶端裝置會使用此資料平面操作來擷取連線至 Greengrass 核心裝置所需的資訊，您可以將它們與 [BatchAssociateClientDeviceWithCoreDevice](#) API 操作建立關聯。當用戶端裝置上線時，可以連線至 AWS IoT Greengrass 雲端服務，並使用 探索 API 來尋找：

- 每個相關聯 Greengrass 核心裝置的 IP 地址和連接埠。
- 核心裝置 CA 憑證，用戶端裝置可用來驗證 Greengrass 核心裝置。

Note

用戶端裝置也可以使用中的探索用戶端 AWS IoT Device SDK 來探索 Greengrass 核心裝置的連線資訊。探索用戶端使用探索 API。如需詳細資訊，請參閱下列內容：

- [測試用戶端裝置通訊](#)
- 《AWS IoT Greengrass Version 1 開發人員指南》中的 [Greengrass 探索 RESTful API](#)。

若要使用此 API 操作，請將 HTTP 請求傳送至 Greengrass 資料平面端點上的探索 API。此 API 端點的格式如下。

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

如需 AWS IoT Greengrass 探索 API 支援 AWS 區域 和端點的清單，請參閱 中的[AWS IoT Greengrass V2 端點和配額](#) AWS 一般參考。此 API 操作僅適用於 Greengrass 資料平面端點。您用來管理元件和部署的控制平面端點與資料平面端點不同。

Note

AWS IoT Greengrass V1 和 的探索 API 相同 AWS IoT Greengrass V2。如果您有連接到 AWS IoT Greengrass V1 核心的用戶端裝置，您可以將它們連接到 AWS IoT Greengrass V2 核心裝置，而無需變用戶端裝置上的程式碼。如需詳細資訊，請參閱《AWS IoT Greengrass Version 1 開發人員指南》中的 [Greengrass 探索 RESTful API](#)。

主題

- [探索身分驗證和授權](#)
- [請求](#)
- [回應](#)
- [使用 cURL 測試探索 API](#)

探索身分驗證和授權

若要使用 探索 API 擷取連線資訊，用戶端裝置必須使用 TLS 交互身分驗證搭配 X.509 用戶端憑證進行身分驗證。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [X.509 用戶端憑證](#)。

用戶端裝置也必須具有執行 `greengrass:Discover` 動作的許可。下列範例 AWS IoT 政策允許名為的 AWS IoT 物件 `Discover` 自行 `MyClientDevice1` 執行。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

```
}  
]  
}
```

Important

核心裝置或 Greengrass 資料平面操作的政策中不支援的 [物件政策變數](#) (iot:Connection.Thing.*). AWS IoT 反之，您可以使用與具有類似名稱的多個裝置相符的萬用字元。例如，您可以指定 MyGreengrassDevice* 來比對 MyGreengrassDevice1、MyGreengrassDevice2 等。

如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的 [AWS IoT Core 政策](#)。

請求

請求包含標準 HTTP 標頭，並傳送至 Greengrass 探索端點，如下列範例所示。

連接埠號碼取決於核心裝置是否設定為透過連接埠 8443 或連接埠 443 傳送 HTTPS 流量。如需詳細資訊，請參閱 [the section called “連線至連接埠 443 或透過網路代理”](#)。

Note

這些範例使用 Amazon Trust Services (ATS) 端點，其可與建議的 ATS 根 CA 憑證搭配使用。端點必須符合根 CA 憑證類型。

連接埠 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/  
thing/thing-name
```

連接埠 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/  
thing/thing-name
```

Note

在連接埠 443 上連線的用戶端必須實作 [Application Layer Protocol Negotiation \(ALPN\)](#) TLS 延伸模組，並在 ProtocolName 中 x-amzn-http-ca 做為傳遞 ProtocolNameList。如需詳細資訊，請參閱《AWS IoT 開發人員指南》中的 [通訊協定](#)。

回應

成功時，回應標頭會包含 HTTP 200 狀態碼，而回應內文會包含探索回應文件。

Note

由於 AWS IoT Greengrass V2 使用與相同的探索 API AWS IoT Greengrass V1，因此回應會根據 AWS IoT Greengrass V1 概念組織資訊，例如 Greengrass 群組。回應包含 Greengrass 群組的清單。在中 AWS IoT Greengrass V2，每個核心裝置都在自己的群組中，其中群組只包含該核心裝置及其連線資訊。

範例 Discover 回應文件

下列文件顯示與一個 Greengrass 核心裝置相關聯之用戶端裝置的回應。核心裝置有一個端點和一個 CA 憑證。

```
{
  "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-----"
  ]
}
]
}

```

下列文件顯示與兩個核心裝置相關聯的用戶端裝置的回應。核心裝置具有多個端點和多個群組 CA 憑證。

```

{
  "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"
            }
          ]
        }
      ]
    },
    {
      "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-----"
    ]
  }
]
}

```

使用 cURL 測試探索 API

如果您 cURL 已安裝，則可以測試 探索 API。下列範例指定用戶端裝置的憑證，以驗證對 Greengrass 探索 API 端點的請求。

```

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

-i 引數指定 輸出 HTTP 回應標頭。您可以使用此選項來協助識別錯誤。

如果請求成功，此命令會輸出類似下列範例的回應。

```

{
  "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"
  ]
}
```

如果命令輸出錯誤，請參閱[故障診斷 Greengrass 探索問題](#)。

在用戶端裝置與 之間轉送 MQTT 訊息 AWS IoT Core

您可以在用戶端裝置與 之間轉送 MQTT 訊息和其他資料 AWS IoT Core。用戶端裝置會連線至核心裝置上執行的 MQTT 代理程式元件。根據預設，核心裝置不會在用戶端裝置與 之間轉送 MQTT 訊息或資料 AWS IoT Core。根據預設，用戶端裝置只能透過 MQTT 彼此通訊。

若要在用戶端裝置和 之間轉送 MQTT 訊息 AWS IoT Core，請將 [MQTT 橋接器元件](#)設定為執行下列動作：

- 將訊息從用戶端裝置轉送至 AWS IoT Core。
- 將訊息從 轉送 AWS IoT Core 至用戶端裝置。

Note

MQTT 橋接器使用 QoS 1 來發佈和訂閱 AWS IoT Core，即使用戶端裝置使用 QoS 0 來發佈和訂閱本機 MQTT 代理程式也一樣。因此，當您從本機 MQTT 代理程式上的用戶端裝置轉送

MQTT 訊息時，您可能會發現額外的延遲 AWS IoT Core。如需核心裝置上 MQTT 組態的詳細資訊，請參閱 [設定 MQTT 逾時和快取設定](#)。

主題

- [設定和部署 MQTT 橋接器元件](#)
- [轉送 MQTT 訊息](#)

設定和部署 MQTT 橋接器元件

MQTT 橋接器元件會使用主題映射清單，每個主題映射都會指定訊息來源和訊息目的地。若要在用戶端裝置與之間轉送訊息 AWS IoT Core，請部署 MQTT 橋接器元件，並在元件組態中指定每個來源和目的地主題。

若要將 MQTT 橋接器元件部署至核心裝置或核心裝置群組，[請建立包含元件的部署](#)。

`aws.greengrass.clientdevices.mqtt.Bridge` 在部署的 MQTT 橋接器元件組態中指定主題映射 `mqttTopicMapping`。

下列範例會定義部署，以設定 MQTT 橋接器元件，在符合用戶端裝置 `clients/+/hello/world` 主題篩選條件的主題上轉送訊息 AWS IoT Core。merge 組態更新需要序列化 JSON 物件。如需詳細資訊，請參閱 [更新元件組態](#)。

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\"}}}"
    }
  }
  ...
}
}

```

轉送 MQTT 訊息

若要在用戶端裝置和 之間轉送 MQTT 訊息 AWS IoT Core，[請設定和部署 MQTT Bridge 元件](#)，並指定要轉送的主題。

Example範例：將主題上的訊息從用戶端裝置轉送至 AWS IoT Core

下列 MQTT 橋接器元件組態指定將符合clients/+/hello/world/event主題篩選條件之主題的訊息從用戶端裝置轉送至 AWS IoT Core。

```

{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}

```

Example範例：將主題上的訊息從 轉送 AWS IoT Core 至用戶端裝置

下列 MQTT 橋接器元件組態指定將符合clients/+/hello/world/event/response主題篩選條件之主題的訊息從 AWS IoT Core 轉送至用戶端裝置。

```

{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}

```

```
}  
}
```

與元件中的用戶端裝置互動

您可以開發與連接到核心裝置的用戶端裝置互動的自訂 Greengrass 元件。例如，您可以開發執行下列動作的元件：

- 從用戶端裝置處理 MQTT 訊息，並將資料傳送至 AWS 雲端目的地。
- 將 MQTT 訊息傳送至用戶端裝置以啟動動作。

用戶端裝置會透過在核心裝置上執行的 MQTT 代理程式元件，連線至核心裝置並與之通訊。根據預設，用戶端裝置只能透過 MQTT 彼此通訊，且 Greengrass 元件無法接收這些 MQTT 訊息或傳送訊息至用戶端裝置。

Greengrass 元件使用[本機發佈/訂閱界面](#)，在核心裝置上進行通訊。若要與 Greengrass 元件中的用戶端裝置通訊，請將[MQTT 橋接器元件](#)設定為執行下列動作：

- 將 MQTT 訊息從用戶端裝置轉送至本機發佈/訂閱。
- 從本機發佈/訂閱用戶端裝置的轉送 MQTT 訊息。

您也可以 Greengrass 元件中與用戶端裝置陰影互動。如需詳細資訊，請參閱[與用戶端裝置影子互動和同步](#)。

主題

- [設定和部署 MQTT 橋接器元件](#)
- [從用戶端裝置接收 MQTT 訊息](#)
- [將 MQTT 訊息傳送至用戶端裝置](#)

設定和部署 MQTT 橋接器元件

MQTT 橋接器元件會使用主題映射清單，每個主題映射都會指定訊息來源和訊息目的地。若要與用戶端裝置通訊，請部署 MQTT 橋接器元件，並在元件組態中指定每個來源和目的地主題。

若要將 MQTT 橋接器元件部署至核心裝置或核心裝置群組，[請建立包含元件的部署](#)。

`aws.greengrass.clientdevices.mqtt.Bridge` 在部署的 MQTT 橋接器元件組態中指定主題映射 `mqttTopicMapping`。

下列範例會定義部署，設定 MQTT 橋接器元件，將clients/MyClientDevice1/hello/world主題從用戶端裝置轉送至本機發佈/訂閱代理程式。merge 組態更新需要序列化 JSON 物件。如需詳細資訊，請參閱[更新元件組態](#)。

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\"}}}"
      }
    }
    ...
  }
}
```

您可以使用 MQTT 主題萬用字元，在符合主題篩選條件的主題上轉送訊息。如果您使用 MQTT 橋接器 v2.2.0 或更新版本，當來源代理程式為本機發佈/訂閱時，您可以在主題篩選條件中使用 MQTT 主題萬用字元。如需詳細資訊，請參閱 [MQTT 橋接器元件組態](#)。

從用戶端裝置接收 MQTT 訊息

您可以訂閱為 MQTT 橋接器元件設定的本機發佈/訂閱主題，以接收來自用戶端裝置的訊息。

從自訂元件中的用戶端裝置接收 MQTT 訊息

1. [設定和部署 MQTT 橋接器元件](#)，從用戶端裝置發佈至本機發佈/訂閱主題的 MQTT 主題轉送訊息。
2. 使用本機發佈/訂閱 IPC 界面來訂閱 MQTT 橋接器轉送訊息的主題。如需詳細資訊，請參閱[發佈/訂閱本機訊息](#)及[SubscribeToTopic](#)。

[Connect and test 用戶端裝置教學](#)課程包含一個區段，您可以在其中開發訂閱用戶端裝置訊息的元件。如需詳細資訊，請參閱[步驟 4：開發與用戶端裝置通訊的元件](#)。

將 MQTT 訊息傳送至用戶端裝置

您可以發佈至您為 MQTT 橋接器元件設定的本機發佈/訂閱主題，以傳送訊息至用戶端裝置。

將 MQTT 訊息發佈至自訂元件中的用戶端裝置

1. [設定和部署 MQTT 橋接器元件](#)，將訊息從本機發佈/訂閱主題轉送至用戶端裝置訂閱的 MQTT 主題。
2. 使用本機發佈/訂閱 IPC 介面發佈至 MQTT 橋接器轉送訊息的主題。如需詳細資訊，請參閱[發佈/訂閱本機訊息](#)及[PublishToTopic](#)。

與用戶端裝置影子互動和同步

您可以使用[影子管理員元件](#)來管理本機影子，包括用戶端裝置影子。您可以使用影子管理員來執行下列動作：

- 在 Greengrass 元件中與用戶端裝置陰影互動。
- 與 同步用戶端裝置影子 AWS IoT Core。

Note

根據 AWS IoT Core 預設，陰影管理員元件不會與 同步陰影。您必須設定影子管理員元件，以指定要同步的用戶端裝置影子。

主題

- [先決條件](#)

- [讓影子管理員與用戶端裝置通訊](#)
- [與元件中的用戶端裝置陰影互動](#)
- [使用 同步用戶端裝置影子 AWS IoT Core](#)

先決條件

若要與用戶端裝置影子互動並同步用戶端裝置影子 AWS IoT Core，核心裝置必須符合下列要求：

- 除了[用戶端裝置支援的 Greengrass 元件之外](#)，核心裝置還必須執行下列元件：
 - [Greengrass 核 v2.6.0 或更新版本](#)
 - [Shadow Manager](#) 2.2.0 版或更新版本
 - [MQTT 橋接器 v2.2.0 或更新版本](#)
- [用戶端裝置驗證](#) 元件必須設定為允許用戶端裝置在[裝置影子主題](#)上進行通訊。

讓影子管理員與用戶端裝置通訊

根據預設，影子管理員元件不會管理用戶端裝置影子。若要啟用此功能，您必須在用戶端裝置和影子管理員元件之間轉送 MQTT 訊息。用戶端裝置使用 MQTT 訊息來接收和傳送裝置陰影更新。影子管理員元件會訂閱本機 Greengrass 發佈/訂閱介面，因此您可以設定 [MQTT 橋接器元件](#)，以在[裝置影子主題](#)上轉送 MQTT 訊息。

MQTT 橋接器元件會使用主題映射清單，每個主題映射都會指定訊息來源和訊息目的地。若要讓影子管理員元件管理用戶端裝置影子，請部署 MQTT 橋接器元件，並指定用戶端裝置影子的影子主題。您必須設定橋接器，在本機 MQTT 和本機發佈/訂閱之間雙向轉送訊息。

若要將 MQTT 橋接器元件部署至核心裝置或核心裝置群組，[請建立包含元件的部署](#)。

`aws.greengrass.clientdevices.mqtt.Bridge` 在部署的 MQTT 橋接器元件組態中指定主題映射 `mqttTopicMapping`。

使用下列範例來設定 MQTT 橋接器元件，以啟用用戶端裝置與影子管理員元件之間的通訊。

Note

您可以在 AWS IoT Greengrass 主控台中使用這些組態範例。如果您使用 AWS IoT Greengrass API，則 `merge` 組態更新需要序列化 JSON 物件，因此必須將下列 JSON 物件序列化為字串。如需詳細資訊，請參閱[更新元件組態](#)。

Example範例：管理所有用戶端裝置影子

下列 MQTT 橋接器組態範例可讓影子管理員管理所有用戶端裝置的所有影子。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+ /shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+ /shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example範例：管理用戶端裝置的陰影

下列 MQTT 橋接器組態範例可讓影子管理員管理名為 之用戶端裝置的所有影子MyClientDevice。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example範例：管理所有用戶端裝置的具名影子

下列 MQTT 橋接器組態範例可讓影子管理員管理所有用戶端裝置名為 DeviceConfiguration的影子。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example範例：管理所有用戶端裝置的未命名影子

下列 MQTT 橋接器組態範例可讓影子管理員管理所有用戶端裝置的未命名影子，但不命名影子。

```
{
  "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"
  }
}
```

與元件中的用戶端裝置陰影互動

您可以開發自訂元件，使用本機影子服務來讀取和修改用戶端裝置的本機影子文件。如需詳細資訊，請參閱[與元件中的陰影互動](#)。

使用 同步用戶端裝置影子 AWS IoT Core

您可以設定影子管理員元件，以與本機用戶端裝置影子狀態同步 AWS IoT Core。如需詳細資訊，請參閱[使用 同步本機裝置陰影 AWS IoT Core](#)。

將 IPv6 用於本機傳訊

您可以設定 IP 偵測器元件使用 IPv6 傳送本機訊息。

Note

您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用 IPv6 傳送本機訊息。

您可以部署 [IP 偵測器元件](#)來偵測和使用 IPv6 地址。您必須更新 IP 偵測器元件的組態，以使用 IPv6 而非 IPv4。如需詳細資訊，請參閱[使用 IP 偵測器自動管理端點](#)。

主題

- [設定 IP 偵測器以使用 IPv6](#)

設定 IP 偵測器以使用 IPv6

如果您有簡單的網路設定，例如與核心裝置位於相同網路上的用戶端裝置，您可以部署 [IP 偵測器元件](#)以使用 IPv6 進行本機傳訊。

IP 偵測器元件會覆寫您手動設定的端點。

⚠ Important

核心裝置的 AWS IoT 政策必須允許使用 IP 偵測器元件的 `greengrass:UpdateConnectivityInfo` 許可。如需詳細資訊，請參閱 [AWS IoT 資料平面操作的 政策及設定 AWS IoT 物件政策](#)。

您可以執行下列任一動作來部署 IP 偵測器元件：

- 使用主控台中的設定探索頁面。如需詳細資訊，請參閱 [設定雲端探索（主控台）](#)。
- 建立和修訂部署以包含 IP 偵測器。您可以使用主控台 AWS CLI、或 AWS API 來管理部署。如需詳細資訊，請參閱 [建立部署](#)。

部署 IP 偵測器元件（主控台）

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇元件。
2. 在元件頁面上，選擇公有元件索引標籤，然後選擇 `aws.greengrass.clientdevices.IPDetector`。
3. 在 `aws.greengrass.clientdevices.IPDetector` 頁面中，選擇部署。
4. 從新增至部署中，選擇要修改的現有部署，或選擇建立新的部署，然後選擇下一步。
5. 如果您選擇建立新的部署，請選擇部署的目標核心裝置或物件群組。在指定目標頁面的部署目標下，選擇核心裝置或物件群組，然後選擇下一步。
6. 在選取元件頁面上，確認已選取 `aws.greengrass.clientdevices.IPDetector` 元件，選擇下一步。
7. 在設定元件頁面上，選取 `aws.greengrass.clientdevices.IPDetector`，然後執行下列動作：
 - a. 選擇設定元件。
 - b. 在設定 `aws.greengrass.clientdevices.IPDetector` 模態、組態更新下，在要合併的組態中，您可以輸入組態更新來設定 IP 偵測器元件。您可以指定下列任何組態選項。將 `includeIPv4Addrs` 設定為 `false`，將 `includeIPv6Addrs` 設定為 `true`。然後，您可以更新其他 IPv6 組態選項。
 - `defaultPort` – （選用）此元件偵測到 IP 地址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。

- `includeIPv4LoopbackAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。
- `includeIPv4LinkLocalAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。
- `includeIPv6LoopbackAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv6 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6LinkLocalAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv6 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv4Addrs` – (選用) 預設值設為 `true`。您可以啟用此選項來發佈核心裝置上找到的 IPv4 地址。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6Addrs` – (選用) 您可以啟用此選項來發佈核心裝置上找到的 IPv6 地址。`includeIPv4Addrs` 設定為 `false` 以使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。

組態更新看起來可能會類似下列範例。

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false,
  "includeIPv6LoopbackAddrs": true,
  "includeIPv6LinkLocalAddrs": true,
  "includeIPv4Addrs": false,
  "includeIPv6Addrs": true
}
```

c. 選擇確認以關閉模態，然後選擇下一步。

8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。

9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署可能需要一分鐘的時間才能完成。

部署 IP 偵測器元件 (AWS CLI)

若要部署 IP 偵測器元件，請建立包含在 components 物件 `aws.greengrass.clientdevices.IPDetector` 中的部署文件，並指定元件的組態更新。遵循中的指示[建立部署](#)建立新的部署或修改現有的部署。

您可以在建立部署文件時指定下列任一選項來設定 IP 偵測器元件：

- `defaultPort` – (選用) 此元件偵測到 IP 地址時要報告的 MQTT 代理程式連接埠。如果您將 MQTT 代理程式設定為使用與預設連接埠 8883 不同的連接埠，則必須指定此參數。
- `includeIPv4LoopbackAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv4 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。
- `includeIPv4LinkLocalAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv4 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。
- `includeIPv6LoopbackAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv6 回送地址。這些是 IP 地址，例如 `localhost`，裝置可以自行通訊。在核心裝置和用戶端裝置在相同系統上執行的測試環境中使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6LinkLocalAddrs` – (選用) 您可以啟用此選項來偵測和報告 IPv6 [連結本機地址](#)。如果核心裝置的網路沒有動態主機組態協定 (DHCP) 或靜態指派的 IP 地址，請使用此選項。您必須將 `includeIPv4Addrs` 設定為 `false`，並將 `includeIPv6Addrs` 設定為 `true`，才能使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv4Addrs` – (選用) 預設值設定為 `true`。您可以啟用此選項來發佈核心裝置上找到的 IPv4 地址。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。
- `includeIPv6Addrs` – (選用) 您可以啟用此選項來發佈核心裝置上找到的 IPv6 地址。`includeIPv4Addrs` 設定為 `false` 以使用此選項。您必須擁有 IP 偵測器 v2.2.0 或更新版本，才能使用此選項。

下列範例部分部署文件指定 使用 IPv6。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

對用戶端裝置進行故障診斷

使用本節中的疑難排解資訊和解決方案，以協助解決 Greengrass 用戶端裝置和用戶端裝置元件的問題。

主題

- [Greengrass 探索問題](#)
- [MQTT 連線問題](#)

Greengrass 探索問題

使用下列資訊來疑難排解 Greengrass 探索的問題。當用戶端裝置使用 [Greengrass 探索 API](#) 來識別其可連線的 Greengrass 核心裝置時，可能會發生這些問題。

主題

- [Greengrass 探索問題 \(HTTP API\)](#)
- [Greengrass 探索問題 \(AWS IoT Device SDK v2 for Python\)](#)
- [Greengrass 探索問題 \(適用於 C++ 的 AWS IoT Device SDK v2\)](#)
- [Greengrass 探索問題 \(適用於 JavaScript 的 AWS IoT Device SDK v2\)](#)
- [Greengrass 探索問題 \(適用於 Java 的 AWS IoT Device SDK v2\)](#)

Greengrass 探索問題 (HTTP API)

使用下列資訊來疑難排解 Greengrass 探索的問題。如果您[使用 cURL 測試探索 API](#)，可能會看到這些錯誤。

主題

- [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

如果您在請求 AWS IoT 中指定非作用中憑證，您可能會看到此錯誤。

檢查用戶端裝置是否具有連接的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[將物件或政策連接至用戶端憑證](#)和[啟用或停用用戶端憑證](#)。

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

如果用戶端裝置沒有greengrass:Discover自行呼叫的許可，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的政策greengrass:Discover。您無法在此許可的Resource區段中使用[物件政策變數](#) (iot:Connection.Thing.*)。如需詳細資訊，請參閱[探索身分驗證和授權](#)。

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組建立關聯。
- 用戶端裝置相關聯的 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組都沒有 MQTT 代理程式端點。
- 用戶端裝置相關聯的 Greengrass 核心裝置都不會執行[用戶端裝置身分驗證元件](#)。

檢查用戶端裝置是否與您要其連線的核心裝置相關聯。然後，檢查核心裝置是否執行[用戶端裝置身分驗證元件](#)，並至少有一個 MQTT 代理程式端點。如需詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)

- [設定雲端探索（主控台）](#)

Greengrass 探索問題 (AWS IoT Device SDK v2 for Python)

使用以下資訊來疑難排解[AWS IoT Device SDK 適用於 Python 的 v2](#) 中 Greengrass 探索的問題。

主題

- [awscli.exceptions.AwsCliError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.](#)
- [awscli.greengrass_discovery.DiscoveryException: \('Error during discover call: response_code=403', 403\)](#)
- [awscli.greengrass_discovery.DiscoveryException: \('Error during discover call: response_code=404', 404\)](#)

awscli.exceptions.AwsCliError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

如果您在請求 AWS IoT 中指定非作用中憑證，您可能會看到此錯誤。

檢查用戶端裝置是否具有連接的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[將物件或政策連接至用戶端憑證](#)和[啟用或停用用戶端憑證](#)。

awscli.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

如果用戶端裝置沒有greengrass:Discover自行呼叫的許可，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的政策greengrass:Discover。您無法在此許可的Resource區段中使用[物件政策變數](#) (iot:Connection.Thing.*)。如需詳細資訊，請參閱[探索身分驗證和授權](#)。

awscli.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組建立關聯。
- 用戶端裝置相關聯的 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組都沒有 MQTT 代理程式端點。

- 用戶端裝置相關聯的 Greengrass 核心裝置都不會執行[用戶端裝置身分驗證元件](#)。

檢查用戶端裝置是否與您要其連線的核心裝置相關聯。然後，檢查核心裝置是否執行[用戶端裝置身分驗證元件](#)，並至少有一個 MQTT 代理程式端點。如需詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)
- [設定雲端探索（主控台）](#)

Greengrass 探索問題（適用於 C++ 的 AWS IoT Device SDK v2）

使用下列資訊對適用於 [AWS IoT Device SDK C++ 的 v2](#) 中的 Greengrass 探索問題進行故障診斷。

主題

- [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.

如果您在請求 AWS IoT 中指定非作用中憑證，您可能會看到此錯誤。

檢查用戶端裝置是否具有連接的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[將物件或政策連接至用戶端憑證](#)和[啟用或停用用戶端憑證](#)。

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

如果用戶端裝置沒有greengrass:Discover自行呼叫的許可，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的政策greengrass:Discover。您無法在此許可的Resource區段中使用[物件政策變數](#) (iot:Connection.Thing.*)。如需詳細資訊，請參閱[探索身分驗證和授權](#)。

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組建立關聯。
- 用戶端裝置相關聯的 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組都沒有 MQTT 代理程式端點。
- 用戶端裝置相關聯的 Greengrass 核心裝置都不會執行[用戶端裝置身分驗證元件](#)。

檢查用戶端裝置是否與您要其連線的核心裝置相關聯。然後，檢查核心裝置是否執行[用戶端裝置身分驗證元件](#)，並至少有一個 MQTT 代理程式端點。如需詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)
- [設定雲端探索（主控台）](#)

Greengrass 探索問題（適用於 JavaScript 的 AWS IoT Device SDK v2）

使用以下資訊對適用於 [AWS IoT Device SDK JavaScript 的 v2](#) 中的 Greengrass 探索問題進行故障診斷。

主題

- [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.

如果您在請求 AWS IoT 中指定非作用中憑證，您可能會看到此錯誤。

檢查用戶端裝置是否具有連接的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[將物件或政策連接至用戶端憑證](#)和[啟用或停用用戶端憑證](#)。

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

如果用戶端裝置沒有greengrass:Discover自行呼叫的許可，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的政策greengrass:Discover。您無法在此許可的Resource區段中使用[物件政策變數](#) (iot:Connection.Thing.*)。如需詳細資訊，請參閱[探索身分驗證和授權](#)。

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組建立關聯。
- 用戶端裝置相關聯的 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組都沒有 MQTT 代理程式端點。
- 用戶端裝置相關聯的 Greengrass 核心裝置都不會執行[用戶端裝置身分驗證元件](#)。

檢查用戶端裝置是否與您要其連線的核心裝置相關聯。然後，檢查核心裝置是否執行[用戶端裝置身分驗證元件](#)，並且至少有一個 MQTT 代理程式端點。如需詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)
- [設定雲端探索（主控台）](#)

Error: Discovery failed (headers: [object Object])

當您執行 Greengrass 探索範例時，您可能會看到此錯誤（沒有 HTTP 回應碼）。此錯誤可能因多種原因而發生。

- 如果用戶端裝置沒有greengrass:Discover自行呼叫的許可，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的政策greengrass:Discover。您無法在此許可的Resource區段中使用[物件政策變數](#) (iot:Connection.Thing.*)。如需詳細資訊，請參閱[探索身分驗證和授權](#)。

- 在下列情況下，您可能會看到此錯誤：
 - 用戶端裝置未與任何 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組建立關聯。
 - 用戶端裝置相關聯的 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組都沒有 MQTT 代理程式端點。
 - 用戶端裝置相關聯的 Greengrass 核心裝置都不會執行[用戶端裝置身分驗證元件](#)。

檢查用戶端裝置是否與您要其連線的核心裝置相關聯。然後，檢查核心裝置是否執行[用戶端裝置身分驗證元件](#)，並至少有一個 MQTT 代理程式端點。如需詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)
- [設定雲端探索（主控台）](#)

Greengrass 探索問題（適用於 Java 的 AWS IoT Device SDK v2）

使用以下資訊來疑難排解適用於 [AWS IoT Device SDK Java 的 v2](#) 中 Greengrass 探索的問題。

主題

- [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.)

如果您在請求 AWS IoT 中指定非作用中憑證，您可能會看到此錯誤。

檢查用戶端裝置是否具有連接的憑證，以及憑證是否處於作用中狀態。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[將物件或政策連接至用戶端憑證](#)和[啟用或停用用戶端憑證](#)。

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

如果用戶端裝置沒有greengrass:Discover自行呼叫的許可，您可能會看到此錯誤。

檢查用戶端裝置的憑證是否具有允許的政策greengrass:Discover。您無法在此許可的Resource區段中使用[物件政策變數](#) (iot:Connection.Thing.*)。如需詳細資訊，請參閱[探索身分驗證和授權](#)。

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

在下列情況下，您可能會看到此錯誤：

- 用戶端裝置未與任何 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組建立關聯。
- 用戶端裝置相關聯的 Greengrass 核心裝置或 AWS IoT Greengrass V1 群組都沒有 MQTT 代理程式端點。
- 用戶端裝置相關聯的 Greengrass 核心裝置都不會執行[用戶端裝置身分驗證元件](#)。

檢查用戶端裝置是否與您要其連線的核心裝置相關聯。然後，檢查核心裝置是否執行[用戶端裝置身分驗證元件](#)，並至少有一個 MQTT 代理程式端點。如需詳細資訊，請參閱下列內容：

- [關聯用戶端裝置](#)
- [管理核心裝置端點](#)
- [設定雲端探索（主控台）](#)

MQTT 連線問題

使用以下資訊對用戶端裝置 MQTT 連線的問題進行故障診斷。當用戶端裝置嘗試透過 MQTT 連線至核心裝置時，可能會發生這些問題。

主題

- [io.moquette.broker.Authorizator: Client does not have read permissions on the topic](#)
- [MQTT 連線問題 \(Python\)](#)
- [MQTT 連線問題 \(C++\)](#)
- [MQTT 連線問題 \(Java\)](#)
- [MQTT 連線問題 \(JavaScript\)](#)

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

當用戶端裝置嘗試訂閱沒有許可的 MQTT 主題時，您可能會在 Greengrass 日誌中看到此錯誤。錯誤訊息包含主題。

檢查[用戶端裝置身分驗證元件](#)的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予主題的 `mqtt:subscribe` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

MQTT 連線問題 (Python)

當您使用 [AWS IoT Device SDK v2 for Python](#) 時，請使用以下資訊對用戶端裝置 MQTT 連線的問題進行故障診斷。

主題

- [AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。

- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

MQTT 連線問題 (C++)

當您將 [AWS IoT Device SDK v2 用於 C++](#) 時，請使用以下資訊對用戶端裝置 MQTT 連線的問題進行故障診斷。

主題

- [AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

`AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred`

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

MQTT 連線問題 (Java)

當您使用[AWS IoT Device SDK 適用於 Java 的 v2](#) 時，請使用以下資訊對用戶端裝置 MQTT 連線的問題進行故障診斷。

主題

- [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`

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)

- [用戶端裝置身分驗證](#)
- [建立部署](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

MQTT 連線問題 (JavaScript)

當您使用 [AWS IoT Device SDK v2 for JavaScript](#) 時，請使用以下資訊對用戶端裝置 MQTT 連線的問題進行故障診斷。

主題

- [AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred](#)
- [AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred](#)

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果[用戶端裝置身分驗證元件](#)未定義授予用戶端裝置連線許可的用戶端裝置授權政策，您可能會看到此錯誤。

檢查用戶端裝置身分驗證元件的組態是否包含下列項目：

- 符合用戶端裝置的裝置群組。
- 該裝置群組的用戶端裝置授權政策，授予用戶端裝置的 `mqtt:connect` 許可。

如需如何部署和設定用戶端裝置身分驗證元件的詳細資訊，請參閱下列內容：

- [設定雲端探索（主控台）](#)
- [用戶端裝置身分驗證](#)
- [建立部署](#)

與裝置陰影互動

Greengrass 核心裝置可以使用元件與[AWS IoT 裝置陰影](#)互動。影子是 JSON 文件，可存放 AWS IoT 物件的目前或所需狀態資訊。無論裝置 AWS IoT 是否連接到其他 AWS IoT Greengrass 元件，陰影都可以讓裝置的狀態變成可用。每個 AWS IoT 裝置都有自己的傳統、未命名陰影。您也可以為每個裝置建立多個具名陰影。

裝置和服務可以使用 MQTT 和預留的 [MQTT 影子主題](#)、[使用 Device Shadow REST API 的 HTTP](#)，以及 [的來建立、更新和刪除雲端影子](#)。<https://docs.aws.amazon.com/iot/latest/developerguide/device-shadow-rest-api.html> [AWS CLI](#)[AWS IoT](#)

[影子管理員](#)元件可讓您的 Greengrass 元件建立、更新和刪除本機影子，方法是使用[本機影子服務](#)和本機發佈/訂閱影子主題。影子管理員也會管理這些本機影子文件在核心裝置上的儲存，並處理影子狀態資訊與雲端影子的同步。

您也可以使用影子管理員元件來管理連線至核心[裝置的用戶端](#)裝置的本機影子。若要讓影子管理員管理用戶端裝置影子，您可以設定 [MQTT 橋接元件](#)，在本機 MQTT 代理程式與本機發佈/訂閱服務之間轉送訊息。如需詳細資訊，請參閱[與用戶端裝置影子互動和同步](#)。

如需 AWS IoT 裝置影子概念的詳細資訊，請參閱《AWS IoT 開發人員指南》中的 [AWS IoT Device Shadow 服務](#)。

主題

- [與元件中的陰影互動](#)
- [使用 同步本機裝置陰影 AWS IoT Core](#)

與元件中的陰影互動

您可以開發自訂元件，包括使用本機影子服務讀取和修改本機影子文件和用戶端裝置影子文件的 Lambda 函數元件。

自訂元件會使用中的 AWS IoT Greengrass 核心 IPC 程式庫與本機影子服務互動 AWS IoT Device SDK。[影子管理員](#)元件會在您的核心裝置上啟用本機影子服務。

若要將影子管理員元件部署到 Greengrass 核心裝置，[請建立包含元件的部署](#)。
`aws.greengrass.ShadowManager`

Note

根據預設，部署陰影管理員元件只會啟用本機陰影操作。若要讓 AWS IoT Greengrass 將核心裝置影子或任何用戶端裝置影子的影子狀態資訊同步至中對應的雲端影子文件 AWS IoT Core，您必須為包含 `synchronize` 參數的影子管理員元件建立組態更新。如需詳細資訊，請參閱[使用 同步本機裝置陰影 AWS IoT Core](#)。

主題

- [擷取和修改陰影狀態](#)
- [對陰影狀態變更做出反應](#)

擷取和修改陰影狀態

影子 IPC 操作會擷取和更新本機影子文件中的狀態資訊。影子管理員元件會在您的核心裝置上處理這些影子文件的儲存。

修改本機陰影狀態

1. 將授權政策新增至自訂元件的配方，以允許元件接收本機影子主題的訊息。

如需授權政策範例，請參閱[本機陰影 IPC 授權政策範例](#)。

2. 使用陰影 IPC 操作來擷取和修改陰影狀態資訊。如需在元件程式碼中使用陰影 IPC 操作的詳細資訊，請參閱[與本機陰影互動](#)。

Note

若要讓核心裝置與用戶端裝置影子互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱[啟用影子管理員以與用戶端裝置通訊](#)。

對陰影狀態變更做出反應

Greengrass 元件使用本機發佈/訂閱界面，在核心裝置上進行通訊。若要讓自訂元件對陰影狀態變更做出反應，您可以訂閱本機發佈/訂閱主題。這可讓元件接收本機影子主題的訊息，然後對這些訊息採取行動。

本機影子主題使用與 AWS IoT 裝置影子 MQTT 主題相同的格式。如需陰影主題的詳細資訊，請參閱《AWS IoT 開發人員指南》中的 [Device Shadow MQTT 主題](#)。

對本機陰影狀態變更做出反應

1. 將存取控制政策新增至自訂元件的配方，以允許元件接收本機影子主題的訊息。

如需授權政策範例，請參閱[本機陰影 IPC 授權政策範例](#)。

2. 若要在元件中啟動自訂動作，請使用 `SubscribeToTopic` IPC 操作來訂閱您要接收訊息的影子主題。如需在元件程式碼中使用本機發佈/訂閱 IPC 操作的詳細資訊，請參閱 [發佈/訂閱本機訊息](#)。
3. 若要叫用 Lambda 函數，請使用事件來源組態來提供影子主題的名稱，並指定它是本機發佈/訂閱主題。如需建立 Lambda 函數元件的詳細資訊，請參閱 [執行 AWS Lambda 函數](#)。

Note

若要让核心裝置與用戶端裝置影子互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱[啟用影子管理員以與用戶端裝置通訊](#)。

使用 同步本機裝置陰影 AWS IoT Core

影子管理員元件可讓 AWS IoT Greengrass 同步本機裝置影子狀態與 AWS IoT Core。您必須修改影子管理員元件的組態，以包含 `synchronization` 組態參數，並指定裝置的 AWS IoT 物件名稱，以及您要同步的影子。

當您設定影子管理員同步影子時，它會同步指定影子的所有狀態變更，無論變更是發生在本機影子文件還是雲端影子文件中。

您也可以指定影子管理員元件是即時還是定期同步影子。根據預設，陰影管理員元件會即時同步陰影，因此核心裝置會在每次更新發生 AWS IoT Core 時傳送和接收陰影更新。您可以設定定期間隔，以減少頻寬用量和費用。

主題

- [先決條件](#)
- [設定陰影管理員元件](#)
- [同步本機陰影](#)

- [陰影合併衝突行為](#)

先決條件

若要與 同步本機影子 AWS IoT Core，您必須設定 Greengrass AWS IoT 核心裝置的政策，以允許下列 AWS IoT Core 影子政策動作。

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

如需詳細資訊，請參閱下列內容：

- 《AWS IoT 開發人員指南》中的[AWS IoT Core 政策動作](#)
- [AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)
- [更新核心裝置的 AWS IoT 政策](#)

設定陰影管理員元件

影子管理員需要影子名稱映射清單，才能將本機影子文件中的影子狀態資訊同步到雲端影子文件中 AWS IoT Core。

若要同步陰影狀態，[請建立包含 元件的部署](#)，並在部署中陰影管理員組態的 `synchronize` 組態參數中指定您要同步的陰影。 `aws.greengrass.ShadowManager`

Note

若要讓核心裝置與用戶端裝置影子互動，您還必須設定和部署 MQTT 橋接器元件。如需詳細資訊，請參閱[啟用影子管理員以與用戶端裝置通訊](#)。

下列範例組態更新會指示影子管理員元件將下列影子與 同步 AWS IoT Core：

- 核心裝置的傳統陰影
- `MyCoreShadow` 為核心裝置命名的
- 名為 `IoT` 物件的傳統影子 `MyDevice2`

- 名為 的影子MyShadowA和名為 MyShadowB的 IoT 物件 MyDevice1

此組態更新會指定 AWS IoT Core 即時同步陰影與 。如果您使用影子管理員 v2.1.0 或更新版本，您可以設定影子管理員元件以定期同步影子。若要設定此功能，請將同步策略變更為 `periodic`，並指定間隔的秒`delay`數。如需詳細資訊，請參閱陰影管理員元件[的策略組態參數](#)。

此組態更新指定 以 和 核心裝置之間的 AWS IoT Core 兩個方向同步陰影。如果您使用影子管理員 v2.2.0 或更新版本，您可以設定影子管理員元件，只同步一個方向的影子。若要設定此功能，請將同步變更為 `directiondeviceToCloud`或 `cloudToDevice`。如需詳細資訊，請參閱陰影管理員元件的[方向組態參數](#)。

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
}
```

同步本機陰影

當 Greengrass 核心裝置連線到 AWS IoT 雲端時，影子管理員會為您在元件組態中指定的影子執行下列任務。行為取決於您指定的陰影同步方向組態選項。根據預設，陰影管理員會使用 `betweenDeviceAndCloud` 選項來同步兩個方向的陰影。如果您使用影子管理員 v2.2.0 或更新版本，您可以設定核心裝置僅以一個方向同步影子，可以是 `cloudToDevice` 或 `deviceToCloud`。

- 如果陰影同步方向組態為 `betweenDeviceAndCloud` 或 `cloudToDevice`，陰影管理員會從 中的雲端陰影文件中擷取報告的狀態資訊 AWS IoT Core。然後，它會更新本機儲存的影子文件，以同步裝置狀態。
- 如果影子同步方向組態為 `betweenDeviceAndCloud` 或 `deviceToCloud`，影子管理員會將裝置的目前狀態發佈至雲端影子文件。

陰影合併衝突行為

在某些情況下，例如核心裝置與網際網路中斷連線時，影子可能會在本機影子服務中和 AWS IoT 雲端中變更，之後影子管理員才會同步變更。因此，本機影子服務與 AWS IoT 雲端之間的所需和報告狀態會有所不同

當影子管理員同步影子時，它會根據下列行為合併變更：

- 如果您使用比 v2.2.0 更舊的影子管理員版本，或當您指定 `betweenDeviceAndCloud` 影子同步方向時，適用下列行為：
 - 當影子所需的狀態發生合併衝突時，影子管理員會使用 AWS IoT 來自雲端的值覆寫本機影子文件的衝突區段。
 - 當影子回報的狀態發生合併衝突時，影子管理員會使用本機影子文件的值覆寫 AWS IoT 雲端中影子的衝突區段。
- 當您指定 `deviceToCloud` 陰影同步方向時，陰影管理員會使用本機陰影文件的值覆寫 AWS IoT 雲端中陰影的衝突區段。
- 當您指定 `cloudToDevice` 陰影同步方向時，陰影管理員會使用來自 AWS IoT 雲端的值覆寫本機陰影文件的衝突區段。

管理 Greengrass 核心裝置上的資料串流

AWS IoT Greengrass 串流管理員可讓您更有效率且更可靠地將大量 IoT 資料傳輸至 AWS 雲端。串流管理員會在將資料串流匯出至之前處理 AWS IoT Greengrass 核心上的資料串流 AWS 雲端。串流管理員會與常見的邊緣案例整合，例如機器學習 (ML) 推論，其中 AWS IoT Greengrass Core 裝置會在將資料匯出至 AWS 雲端 或本機儲存目的地之前處理和分析資料。

串流管理員提供通用界面，可簡化自訂元件開發，因此您不需要建置自訂串流管理功能。您的元件可以使用標準化機制來處理大量串流和管理本機資料保留政策。您可以為每個串流定義儲存類型、大小和資料保留的政策，以控制串流管理員處理和匯出資料的方式。

串流管理員可在間歇性或有限連線的環境中運作。您可以定義頻寬使用、逾時行為，以及 AWS IoT Greengrass 核心在連線或中斷連線時處理串流資料的方式。您也可以設定優先順序，以控制核心匯出串流至的順序 AWS IoT Greengrass AWS 雲端。這可讓您比其他資料更快地處理關鍵資料。

您可以設定串流管理員自動將資料匯出至，AWS 雲端 以供儲存或進一步處理和分析。串流管理員支援匯出至下列 AWS 雲端 目的地：

- 您針對資料執行進階分析的頻道 in AWS IoT Analytics. AWS IoT Analytics let，以協助做出商業決策並改善機器學習模型。如需詳細資訊，請參閱《AWS IoT Analytics使用者指南》中的[什麼是AWS IoT Analytics ?](#)。
- Amazon Kinesis Data Streams 中的串流。您可以使用 Kinesis Data Streams 彙總大量資料，並將其載入資料倉儲或 MapReduce 叢集。如需詳細資訊，請參閱《Amazon Kinesis Data Streams 開發人員指南》中的[什麼是 Amazon Kinesis Data Streams ?](#)。
- 您大規模從工業設備收集、組織和分析資料的資產屬性 in. AWS IoT SiteWise AWS IoT SiteWise let。如需詳細資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[什麼是 AWS IoT SiteWise ?](#)。
- Amazon Simple Storage Service Amazon S3 中的物件。您可以使用 Amazon S3 來存放和擷取大量資料。如需詳細資訊，請參閱[《Amazon Simple Storage Service 開發人員指南》中的什麼是 Amazon S3 ?](#)。

串流管理工作流程

您的 IoT 應用程式會透過 Stream Manager SDK 與串流管理員互動。

在簡單的工作流程中，AWS IoT Greengrass 核心上的元件會使用 IoT 資料，例如時間序列溫度和壓力指標。元件可能會篩選或壓縮資料，然後呼叫 Stream Manager SDK，將資料寫入串流管理員中的

串流。串流管理員可以根據您為串流定義的政策，AWS 雲端 自動將串流匯出至 。元件也可以將資料直接傳送到本機資料庫或儲存儲存庫。

您的 IoT 應用程式可以包含讀取或寫入串流的多個自訂元件。這些元件可以讀取和寫入串流，以篩選、彙總和分析 AWS IoT Greengrass 核心裝置上的資料。這可讓您快速回應本機事件，並在資料從核心傳輸到 AWS 雲端 或本機目的地之前擷取寶貴的資訊。

若要開始使用，請將串流管理員元件部署到您的 AWS IoT Greengrass 核心裝置。在 部署中，設定串流管理員元件參數，以定義套用至 Greengrass 核心裝置上所有串流的設定。使用這些參數來根據您的業務需求和環境限制，控制串流管理員如何存放、處理和匯出串流。

設定串流管理員之後，您可以建立和部署 IoT 應用程式。這些通常是在 Stream Manager SDK `StreamManagerClient` 中用來建立串流並與串流互動的自訂元件。建立串流時，您可以定義每個串流政策，例如匯出目的地、優先順序和持久性。

要求

下列需求適用於使用串流管理員：

- 除了 AWS IoT Greengrass Core 軟體之外，串流管理員至少需要 70 MB RAM。您的總記憶體需求視您的工作負載而定。
- AWS IoT Greengrass 元件必須使用 Stream Manager SDK 與串流管理員互動。串流管理員 SDK 提供下列語言：
 - [適用於 Java 的 Stream Manager SDK](#) (1.1.0 版或更新版本)
 - [適用於 Node.js 的 Stream Manager SDK](#) (v1.1.0 或更新版本)
 - [適用於 Python 的 Stream Manager SDK](#) (1.1.0 版或更新版本)
- AWS IoT Greengrass 元件必須在其配方中指定串流管理員元件 (`aws.greengrass.StreamManager`) 做為相依性，才能使用串流管理員。

Note

如果您使用串流管理員將資料匯出至雲端，則無法將串流管理員元件的 2.0.7 版升級至 v2.0.8 和 v2.0.11 之間的版本。如果您是第一次部署串流管理員，強烈建議您部署最新版本的串流管理員元件。

- 如果您定義串流的 AWS 雲端 匯出目的地，則必須建立匯出目標，並在 [Greengrass 裝置角色](#) 中授予存取許可。視目的地而定，可能也會套用其他要求。如需詳細資訊，請參閱：

- [the section called “AWS IoT Analytics 頻道”](#)
- [the section called “Amazon Kinesis 資料串流”](#)
- [the section called “AWS IoT SiteWise 資產屬性”](#)
- [the section called “Amazon S3 物件”](#)

您負責維護這些 AWS 雲端 資源。

資料安全

當您使用串流管理員時，請注意下列安全性考量。

本機資料安全性

AWS IoT Greengrass 不會在核心裝置上的本機元件之間加密靜態或傳輸中的串流資料。

- 靜態資料。串流資料以本機之方式儲存在儲存目錄中。為了資料安全，如果啟用，AWS IoT Greengrass 則依賴檔案許可和全磁碟加密。您可以使用選用的 [STREAM_MANAGER_STORE_ROOT_DIR](#) 參數來指定儲存目錄。如果您稍後將此參數變更為使用不同的儲存目錄，AWS IoT Greengrass 不會刪除先前的儲存目錄或其內容。
- 本機傳輸中的資料。AWS IoT Greengrass 不會加密資料來源、AWS IoT Greengrass 元件、串流管理員 SDK 和串流管理員之間本機傳輸中的串流資料。
- 傳輸到的資料 AWS 雲端。串流管理員匯出至 的資料串流 AWS 雲端 使用標準 AWS 服務用戶端加密搭配 Transport Layer Security (TLS)。

用戶端身分驗證

串流管理員用戶端使用串流管理員 SDK 與串流管理員通訊。啟用用戶端身分驗證時，只有 Greengrass 元件可以與串流管理員中的串流互動。停用用戶端身分驗證時，在 Greengrass 核心裝置上執行的任何程序都可以與串流管理員中的串流互動。您應該只在商業案例需要時，才停用身分驗證。

您可以使用 [STREAM_MANAGER_AUTHENTICATE_CLIENT](#) 參數來設定用戶端身分驗證模式。您可以在將串流管理員元件部署至核心裝置時設定此參數。

	已啟用	Disabled
參數值	true (預設和建議)	false

	已啟用	Disabled
允許用戶端	核心裝置上的 Greengrass 元件	核心裝置上的 Greengrass 元件 在 Greengrass 核心裝置上執行的其他程序

另請參閱

- [the section called “設定串流管理員”](#)
- [the section called “使用 StreamManagerClient 搭配串流”](#)
- [the section called “支援雲端目的地的匯出組態”](#)

設定 AWS IoT Greengrass 串流管理員

在 Greengrass 核心裝置上，串流管理員可以存放、處理和匯出 IoT 裝置資料。串流管理員提供您用來設定執行時間設定的參數。這些設定適用於 Greengrass 核心裝置上的所有串流。您可以在部署元件時，使用 AWS IoT Greengrass 主控台或 API 來設定串流管理員設定。變更會在部署完成後生效。

串流管理員參數

串流管理員提供下列參數，您可以在將元件部署到核心裝置時設定這些參數。所有參數都是選用的。

儲存目錄

參數名稱：STREAM_MANAGER_STORE_ROOT_DIR

用於存放串流之本機資料夾的絕對路徑。此值必須以正斜線開頭 (例如 /data)。

您必須指定現有的資料夾，而且[執行串流管理員元件的系統使用者](#)必須具有讀取和寫入此資料夾的許可。例如，您可以執行下列命令來建立和設定資料夾 /var/greengrass/streams，您指定此資料夾做為串流管理員根資料夾。這些命令允許預設系統使用者 ggc_user 讀取和寫入此資料夾。

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

如需保護串流資料安全的資訊，請參閱 [the section called “本機資料安全性”](#)。

預設：`/greengrass/v2/work/aws.greengrass.StreamManager`

伺服器連接埠

參數名稱：STREAM_MANAGER_SERVER_PORT

用於與串流管理員通訊的本機連接埠號碼。預設值為 8088。

您可以指定 0 使用隨機的可用連接埠。

驗證用戶端

參數名稱：STREAM_MANAGER_AUTHENTICATE_CLIENT

表示用戶端是否須經過驗證才能與串流管理員互動。用戶端和串流管理員之間的所有互動都由串流管理員 SDK 控制。此參數決定哪些用戶端可以呼叫 Stream Manager SDK 來使用串流。如需詳細資訊，請參閱[the section called “用戶端身分驗證”](#)。

有效值為 true 或 false。預設值為 true (建議)。

- true。僅允許 Greengrass 元件做為用戶端。元件使用內部 AWS IoT Greengrass 核心通訊協定來驗證 Stream Manager SDK。
- false。允許在 AWS IoT Greengrass Core 上執行的任何程序成為用戶端。除非您的商業案例需要，false 否則請勿將值設定為。例如，false 只有在核心裝置上的非元件程序必須直接與串流管理員通訊時，才能使用。

最高頻寬

參數名稱：STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH

可用來匯出資料的平均最高頻寬 (以千位元數/秒為單位)。預設允許無限使用可用頻寬。

執行緒集區大小

參數名稱：STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE

可用於匯出資料的作用中執行緒數量上限。預設值為 5。

最佳大小取決於您的硬體、串流磁碟區和規劃的匯出串流數量。如果匯出速度很慢，您可以調整此設定，找出適合您硬體和商務案例的最佳大小。核心裝置硬體的 CPU 和記憶體是限制因素。首先，您可以嘗試將此值設定為等同於裝置上處理器核心的數量。

請小心不要設定高於硬體可支援的大小。每個串流都會耗用硬體資源，因此請嘗試限制受限裝置上的匯出串流數量。

JVM 引數

參數名稱：JVM_ARGS

自訂 Java 虛擬機器參數，以在啟動時傳遞給串流管理員。多個引數應用空格分隔。

僅限必須覆寫 JVM 使用的預設設定時，才能使用此參數。例如，如果您計劃匯出大量串流，可能需要增加預設堆積大小。

Logging level (記錄層級)

參數名稱：LOG_LEVEL

元件的記錄層級。從下列日誌層級中進行選擇，此處依層級順序列出：

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

預設：INFO

分段上傳的大小下限

參數名

稱：STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTE

分段中部分的大小下限（以位元組為單位）上傳至 Amazon S3。串流管理員使用此設定和輸入檔案的大小，來判斷如何在分段 PUT 請求中批次處理資料。預設和最小值為5242880位元組 (5 MB)。

Note

串流管理員使用串流的 `sizeThresholdForMultipartUploadBytes` 屬性來判斷要匯出至 Amazon S3，做為單一或分段上傳。使用者定義的 Greengrass 元件會在建立匯出至 Amazon S3 的串流時設定此閾值。預設閾值為 5 MB。

另請參閱

- [管理 Greengrass 核心裝置上的資料串流](#)

- [使用 StreamManagerClient 搭配串流](#)
- [匯出支援 AWS 雲端 目的地的組態](#)

建立使用串流管理員的自訂元件

在自訂 Greengrass 元件中使用串流管理員來存放、處理和匯出 IoT 裝置資料。使用本節中的程序和範例來建立與串流管理員搭配使用的元件配方、成品和應用程式。如需如何開發和測試元件的詳細資訊，請參閱 [建立 AWS IoT Greengrass 元件](#)。

主題

- [定義使用串流管理員的元件配方](#)
- [連線至應用程式程式碼中的串流管理員](#)

定義使用串流管理員的元件配方

若要在自訂元件中使用串流管理員，您必須將 `aws.greengrass.StreamManager` 元件定義為相依性。您也必須提供 Stream Manager SDK。完成下列任務，以您選擇的語言下載和使用 Stream Manager SDK。

使用適用於 Java 的 Stream Manager 開發套件

適用於 Java 的 Stream Manager 開發套件以 JAR 檔案提供，您可以用來編譯元件。然後，您可以建立包含 Stream Manager SDK 的應用程式 JAR、將應用程式 JAR 定義為元件成品，並在元件生命週期中執行應用程式 JAR。

使用適用於 Java 的 Stream Manager 開發套件

1. 下載適用於 [Java JAR 的 Stream Manager SDK 檔案](#)。
2. 執行下列其中一項操作，從 Java 應用程式和 Stream Manager SDK JAR 檔案建立元件成品：
 - 將應用程式建置為包含 Stream Manager SDK JAR 的 JAR 檔案，然後在元件配方中執行此 JAR 檔案。
 - 將 Stream Manager SDK JAR 定義為元件成品。當您在元件配方中執行應用程式時，請將該成品新增至 classpath。

您的元件配方可能如下所示。此元件會執行 [StreamManagerS3.java](#) 範例的修改版本，其中 `StreamManagerS3.jar` 包含 Stream Manager SDK JAR。

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
```

如需如何開發和測試元件的詳細資訊，請參閱 [建立 AWS IoT Greengrass 元件](#)。

使用適用於 Python 的 Stream Manager SDK

適用於 Python 的 Stream Manager SDK 可作為原始程式碼提供，您可以包含在元件中。建立 Stream Manager SDK 的 ZIP 檔案、將 ZIP 檔案定義為元件成品，並在元件生命週期中安裝 SDK 的需求。

使用適用於 Python 的 Stream Manager SDK

1. 複製或下載 [aws-greengrass-stream-manager-sdk-python](#) 儲存庫。

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-python.git
```

2. 建立包含 `stream_manager` 資料夾的 ZIP 檔案，其中包含適用於 Python 的 Stream Manager SDK 的原始程式碼。您可以提供此 ZIP 檔案做為元件成品，讓 AWS IoT Greengrass Core 軟體在安裝元件時解壓縮。請執行下列操作：

- a. 開啟資料夾，其中包含您在上一個步驟中複製或下載的儲存庫。

```
cd aws-greengrass-stream-manager-sdk-python
```

- b. 將 `stream_manager` 資料夾壓縮至名為 `stream_manager_sdk.zip` 的 ZIP 檔案。

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. 確認 `stream_manager_sdk.zip` 檔案包含 `stream_manager` 資料夾及其內容。執行下列命令來列出 ZIP 檔案的內容。

Linux or Unix

```
unzip -l stream_manager_sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream_manager_sdk.zip
```

輸出應看起來如下列內容。

```
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. 將 Stream Manager SDK 成品複製到元件的成品資料夾。除了 Stream Manager SDK ZIP 檔案，您的元件會使用 `SDK requirements.txt` 的檔案來安裝 Stream Manager SDK 的相依性。將 `~/greengrass-components` 取代為您用於本機開發之資料夾的路徑。

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. 建立您的元件配方。在配方中，執行下列動作：
 - a. 將 `stream_manager_sdk.zip` 和 `requirements.txt` 定義為成品。
 - b. 將 Python 應用程式定義為成品。
 - c. 在安裝生命週期中，從安裝 Stream Manager SDK 需求 `requirements.txt`。
 - d. 在執行生命週期中，將 Stream Manager SDK 附加至 `PYTHONPATH`，然後執行您的 Python 應用程式。

您的元件配方可能如下所示。此元件會執行 [stream_manager_s3.py](#) 範例。

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
```

如需如何開發和測試元件的詳細資訊，請參閱 [建立 AWS IoT Greengrass 元件](#)。

使用適用於 JavaScript 的 Stream Manager SDK

適用於 JavaScript 的 Stream Manager SDK 可作為原始程式碼提供，您可以包含在元件中。建立 Stream Manager SDK 的 ZIP 檔案、將 ZIP 檔案定義為元件成品，並在元件生命週期中安裝 SDK。

使用適用於 JavaScript 的 Stream Manager SDK

1. 複製或下載 [aws-greengrass-stream-manager-sdk-js](#) 儲存庫。

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
```

2. 建立包含 aws-greengrass-stream-manager-sdk 資料夾的 ZIP 檔案，其中包含適用於 JavaScript 的 Stream Manager SDK 的原始程式碼。您可以提供此 ZIP 檔案做為元件成品，讓 AWS IoT Greengrass Core 軟體在安裝元件時解壓縮。請執行下列操作：

- a. 開啟資料夾，其中包含您在上一個步驟中複製或下載的儲存庫。

```
cd aws-greengrass-stream-manager-sdk-js
```

- b. 將aws-greengrass-stream-manager-sdk資料夾壓縮至名為的 ZIP 檔案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. 確認 `stream-manager-sdk.zip` 檔案包含 `aws-greengrass-stream-manager-sdk` 資料夾及其內容。執行下列命令來列出 ZIP 檔案的內容。

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

輸出應看起來如下列內容。

```
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. 將 Stream Manager SDK 成品複製到元件的成品資料夾。將 `~/greengrass-components` 取代為您用於本機開發之資料夾的路徑。

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. 建立您的元件配方。在配方中，執行下列動作：
 - a. stream-manager-sdk.zip 定義為成品。
 - b. 將 JavaScript 應用程式定義為成品。
 - c. 在安裝生命週期中，從stream-manager-sdk.zip成品安裝 Stream Manager SDK。此npm install命令會建立node_modules資料夾，其中包含 Stream Manager SDK 及其相依性。
 - d. 在執行生命週期中，將 node_modules 資料夾附加至 NODE_PATH，然後執行 JavaScript 應用程式。

您的元件配方可能如下所示。此元件會執行 [StreamManagerS3](#) 範例。

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
```


如需如何開發和測試元件的詳細資訊，請參閱 [建立 AWS IoT Greengrass 元件](#)。

連線至應用程式程式碼中的串流管理員

若要連線至應用程式中的串流管理員，StreamManagerClient請從串流管理員 SDK 建立 執行個體。此用戶端會連線至其預設連接埠 8088 上的串流管理員元件，或您指定的連接埠。如需建立執行個體StreamManagerClient後如何使用的詳細資訊，請參閱 [使用 StreamManagerClient 搭配串流](#)。

Example範例：使用預設連接埠連線至串流管理員

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範例：使用非預設連接埠連線至串流管理員

如果您使用預設以外的連接埠設定串流管理員，則必須使用[程序間通訊](#)從元件組態擷取連接埠。

Note

port 組態參數包含您在部署串流管理員STREAM_MANAGER_SERVER_PORT時在 中指定的值。

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
    IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCCClient greengrassCoreIPCCClient = new
    GreengrassCoreIPCCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
        greengrassCoreIPCCClient.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.
```

使用 StreamManagerClient 搭配串流

在 Greengrass 核心裝置上執行的使用者定義 Greengrass 元件可以使用 Stream Manager SDK 中的 StreamManagerClient 物件，在串流[管理員](#)中建立串流，然後與串流互動。當元件建立串流時，它會定義串流的 AWS 雲端目的地、優先順序和其他匯出和資料保留政策。若要將資料傳送至串流管理員，元件會將資料附加至串流。如果為串流定義匯出目的地，串流管理員會自動匯出串流。

Note

一般而言，串流管理員的用戶端是使用者定義的 Greengrass 元件。如果您的商業案例需要，您也可以允許在 Greengrass 核心（例如 Docker 容器）上執行的非元件程序與串流管理員互動。如需詳細資訊，請參閱[the section called “用戶端身分驗證”](#)。

本主題中的程式碼片段會示範用戶端如何呼叫 `StreamManagerClient` 方法以使用串流。如需方法及其引數的實作詳細資訊，請使用每個程式碼片段後列出的 SDK 參考連結。

如果您在 Lambda 函數中使用串流管理員，Lambda 函數應該在函數處理常式 `StreamManagerClient` 之外執行個體化。如果在處理常式內執行個體化，則該函數在每次叫用時都會建立 `client` 以及與串流管理員的連線。

Note

如果您在處理常式內將 `StreamManagerClient` 執行個體化，則必須在 `client` 完成其工作時明確呼叫 `close()` 方法。否則，`client` 會將連線保持在開啟狀態，並將另一個執行緒保持在執行狀態，直到指令碼結束為止。

`StreamManagerClient` 支援下列操作：

- [the section called “建立訊息串流”](#)
- [the section called “附加訊息”](#)
- [the section called “讀取訊息”](#)
- [the section called “列出串流”](#)
- [the section called “描述訊息串流”](#)
- [the section called “更新訊息串流”](#)
- [the section called “刪除訊息串流”](#)

建立訊息串流

若要建立串流，使用者定義的 Greengrass 元件會呼叫建立方法並傳入 `MessageStreamDefinition` 物件。此物件會指定串流的唯一名稱，並定義串流管理員在達到串

流大小上限時應如何處理新資料。您可以使用 `MessageStreamDefinition` 及其資料類型 (例如 `ExportDefinition`、`StrategyOnFull` 和 `Persistence`) 來定義其他串流屬性。其中包含：

- 自動匯出的目標 AWS IoT SiteWise、AWS IoT Analytics Kinesis Data Streams 和 Amazon S3 目的地。如需詳細資訊，請參閱[the section called “支援雲端目的地的匯出組態”](#)。
- 匯出優先順序。串流管理員會先匯出優先順序較高的串流，然後再匯出較低的串流。
- AWS IoT Analytics、Kinesis Data Streams 和 AWS IoT SiteWise 目的地的批次大小上限和批次間隔。符合任一條件時，串流管理員會匯出訊息。
- 存留時間 (TTL)。保證串流資料可供處理的時間量。您應該確定資料可以在這段期間內使用。這不是刪除政策。TTL 期間後，資料可能不會立即刪除。
- 串流持久性。選擇此選項可將串流儲存至檔案系統，以便在核心重新啟動期間保留資料，或將串流儲存在記憶體中。
- 起始序號。指定要用作匯出中起始訊息的訊息序號。

如需的詳細資訊 `MessageStreamDefinition`，請參閱目標語言的 SDK 參考：

- Java 開發套件中的 [MessageStreamDefinition](#)
- Node.js SDK 中的 [MessageStreamDefinition](#)
- Python SDK 中的 [MessageStreamDefinition](#)

Note

`StreamManagerClient` 也提供目標目的地，可用來將串流匯出至 HTTP 伺服器。此目標僅供測試之用。它不穩定或不支援在生產環境中使用。

建立串流後，您的 Greengrass 元件可以將[訊息附加](#)到串流，以傳送資料進行匯出，並從串流[讀取訊息](#)以進行本機處理。您建立的串流數量取決於您的硬體功能和商業案例。其中一個策略是為 AWS IoT Analytics 或 Kinesis 資料串流中的每個目標頻道建立串流，但您可以定義串流的多個目標。串流的生命週期相當耐久。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

以下程式碼片段會建立名為 `StreamName` 的串流。它定義 `MessageStreamDefinition` 和次級資料類型中的串流屬性。

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 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.
```

Python SDK 參考：[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 AWS ##.
            new ExportDefinition()
                .withKinesis(null)
                .withIotAnalytics(null)
                .withIotSitewise(null)
                .withS3(null)
            )
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Java 開發套件參考：[createMessageStream](#) | [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 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.  
});
```

Node.js SDK 參考：[createMessageStream](#) | [MessageStreamDefinition](#)

如需設定匯出目的地的詳細資訊，請參閱 [the section called “支援雲端目的地的匯出組態”](#)。

附加訊息

若要將資料傳送至串流管理員以進行匯出，您的 Greengrass 元件會將資料附加至目標串流。匯出目的地會決定要傳遞至此方法的資料類型。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

AWS IoT Analytics 或 Kinesis Data Streams 匯出目的地

下列程式碼片段附加一個訊息到名為 `StreamName` 的串流。對於 AWS IoT Analytics 或 Kinesis Data Streams 目的地，您的 Greengrass 元件會附加資料的 Blob。

此程式碼片段有下列需求：

- 最低串流管理員 SDK 版本：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.
```

Python SDK 參考：[end_message](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
    array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[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.
});
```

Node.js SDK 參考：[appendMessage](#)

AWS IoT SiteWise 匯出目的地

下列程式碼片段附加一個訊息到名為 `StreamName` 的串流。對於 AWS IoT SiteWise 目的地，您的 Greengrass 元件會附加序列化 `PutAssetPropertyValueEntry` 物件。如需詳細資訊，請參閱 [the section called “匯出至 AWS IoT SiteWise”](#)。

Note

當您傳送資料至 AWS IoT SiteWise，您的資料必須符合 `BatchPutAssetPropertyValue` 動作的要求。如需詳細資訊，請參閱《AWS IoT SiteWise API 參考》中的 [BatchPutAssetPropertyValue](#)。

此程式碼片段有下列需求：

- 最低串流管理員 SDK 版本：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.

```

Python SDK 參考：[end_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.
}

```

Java 開發套件參考：[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.
});

```

Node.js SDK 參考：[appendMessage](#) | [PutAssetPropertyValueEntry](#)

Amazon S3 匯出目的地

下列程式碼片段會將匯出任務附加至名為 `StreamName` 的串流。對於 Amazon S3 目的地，您的 Greengrass 元件會附加序列化 `S3ExportTaskDefinition` 物件，其中包含來源輸入檔案和目標 Amazon S3 物件的相關資訊。如果指定的物件不存在，串流管理員會為您建立它。如需詳細資訊，請參閱 [the section called “匯出至 Amazon S3”](#)。

此程式碼片段有下列需求：

- 最低串流管理員 SDK 版本：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.
```

Python SDK 參考：[end_message](#) | [S3ExportTaskDefinition](#)

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.
```

```
}
```

Java 開發套件參考：[appendMessage](#) | [S3ExportTaskDefinition](#)

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.
});
```

Node.js SDK 參考：[appendMessage](#) | [S3ExportTaskDefinition](#)

讀取訊息

從串流讀取訊息。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

下列程式碼片段可從名為 StreamName 的串流讀取訊息。讀取方法需要一個選用的 ReadMessagesOptions 物件以指定序號，從要讀取的最小、最大數字和讀取訊息的逾時開始讀取。

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_message_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.

```

Python SDK 參考：[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_message_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.
}

```

Java 開發套件參考：[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.  
});
```

Node.js SDK 參考：[readMessages](#) | [ReadMessagesOptions](#)

列出串流

取得串流管理員中的串流清單。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

下列程式碼片段可獲取串流管理員中的串流清單 (依據名稱)。

Python

```
client = StreamManagerClient()  
  
try:  
    stream_names = client.list_streams()  
except StreamManagerException:  
    pass  
    # Properly handle errors.  
except ConnectionError or asyncio.TimeoutError:  
    pass  
    # Properly handle errors.
```

Python SDK 參考：[list_streams](#)

Java

```
try (final StreamManagerClient client =  
    StreamManagerClientFactory.standard().build()) {  
    List<String> streamNames = client.listStreams();  
} catch (StreamManagerException e) {
```

```
// Properly handle exception.  
}
```

Java 開發套件參考：[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.  
});
```

Node.js SDK 參考：[listStreams](#)

描述訊息串流

取得串流的中繼資料，包括串流定義、大小和匯出狀態。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

下列程式碼片段可獲取名為 StreamName 的串流相關的中繼資料，包括串流定義、大小和匯出工具狀態。

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.

```

Python SDK 參考：[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.
}

```

Java 開發套件參考：[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.
});
```

Node.js SDK 參考：[describeMessageStream](#)

更新訊息串流

更新現有串流的屬性。如果您的需求在建立串流之後有所變更，建議您更新串流。例如：

- 新增 AWS 雲端 目的地的新[匯出組態](#)。
- 增加串流的大小上限，以變更資料匯出或保留的方式。例如，串流大小結合您在完整設定上的策略，可能會導致資料遭到刪除或拒絕，然後串流管理員才能處理資料。
- 暫停和繼續匯出；例如，如果匯出任務長時間執行，而且您想要對上傳資料進行評價。

您的 Greengrass 元件遵循此高階程序來更新串流：

1. [取得串流的描述。](#)

2. 更新對應 `MessageStreamDefinition` 和次級物件的目標屬性。
3. 在更新的 中傳遞 `MessageStreamDefinition`。請務必包含更新串流的完整物件定義。未定義的屬性會還原為預設值。

您可以指定要用作匯出中起始訊息的訊息序號。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

下列程式碼片段會更新名為 的串流 `StreamName`。它會更新匯出至 Kinesis Data Streams 之串流的多個屬性。

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.
```

Python SDK 參考：[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 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.
}
```

Java 開發套件參考：[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 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.
    });

```

Node.js SDK 參考：[updateMessageStream](#) | [MessageStreamDefinition](#)

更新串流的限制

更新串流時，適用下列限制條件。除非以下清單中另有說明，否則更新會立即生效。

- 您無法更新串流的持久性。若要變更此行為，[請刪除串流並建立定義新持久性政策的串流](#)。
- 您只能在下列情況下更新串流的大小上限：

- 大小上限必須大於或等於串流的目前大小。若要尋找此資訊，[請描述串流](#)，然後檢查傳回 `MessageStreamInfo` 物件的儲存狀態。
- 大小上限必須大於或等於串流的區段大小。
- 您可以將串流區段大小更新為小於串流大小上限的值。更新的設定會套用至新的客群。
- 存留時間 (TTL) 屬性的更新適用於新的附加操作。如果您減少此值，串流管理員也可能會刪除超過 TTL 的現有區段。
- 完整屬性策略的更新適用於新的附加操作。如果您設定策略來覆寫最舊的資料，串流管理員也可能會根據新設定覆寫現有的區段。
- 寫入屬性排清的更新會套用至新訊息。
- 匯出組態的更新會套用至新的匯出。更新請求必須包含您要支援的所有匯出組態。否則，串流管理員會刪除它們。
 - 當您更新匯出組態時，請指定目標匯出組態的識別符。
 - 若要新增匯出組態，請指定新匯出組態的唯一識別符。
 - 若要刪除匯出組態，請省略匯出組態。
- 若要[更新](#)串流中匯出組態的起始序號，您必須指定小於最新序號的值。若要尋找此資訊，[請描述串流](#)，然後檢查傳回 `MessageStreamInfo` 物件的儲存狀態。

刪除訊息串流

刪除串流。刪除串流時，磁碟中該串流的所有儲存資料都會刪除。

要求

此操作有下列需求：

- 最低串流管理員 SDK 版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

範例

下列程式碼片段會刪除名為 `StreamName` 的串流。

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.
```

Python SDK 參考：[deleteMessageStream](#)

Java

```
try (final StreamManagerClient client =
    StreamManagerClientFactory.standard().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 開發套件參考：[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.
});
```

Node.js SDK 參考：[deleteMessageStream](#)

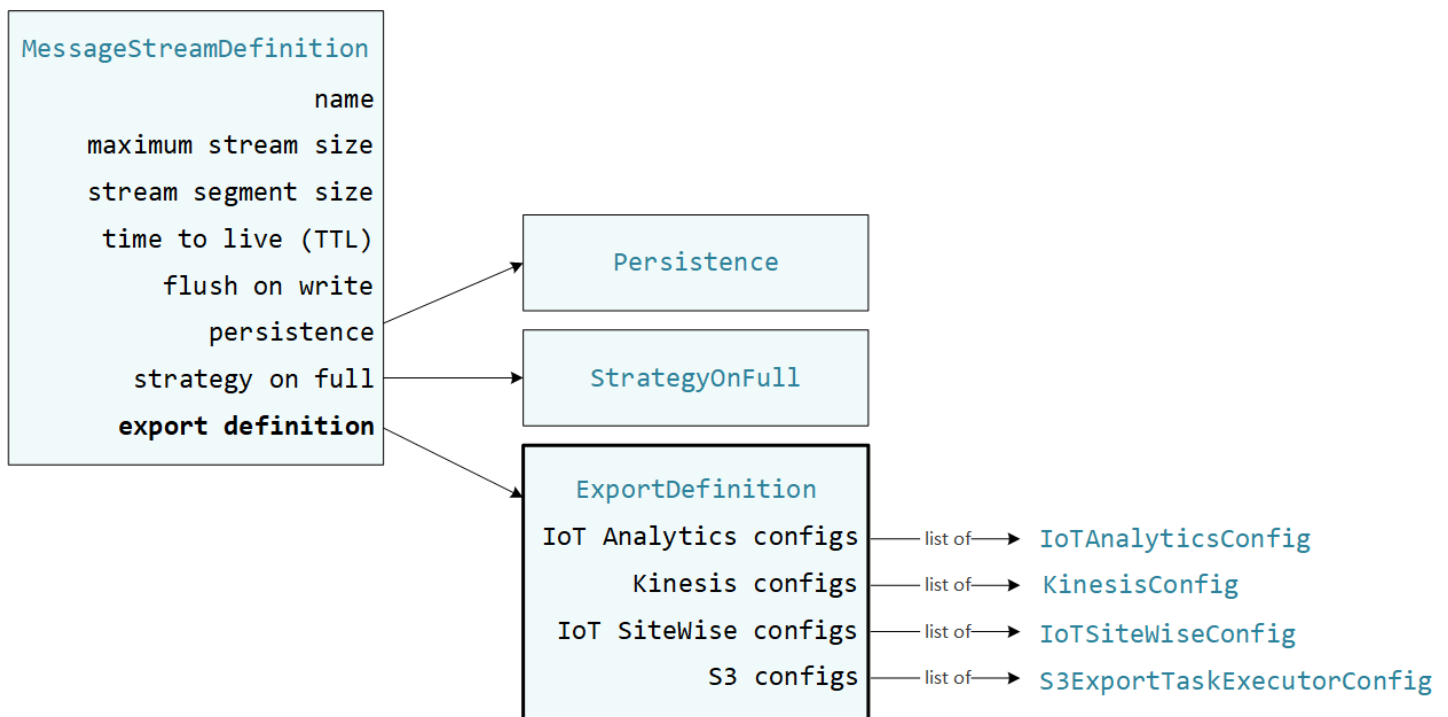
另請參閱

- [管理 Greengrass 核心裝置上的資料串流](#)

- [設定 AWS IoT Greengrass 串流管理員](#)
- [匯出支援 AWS 雲端 目的地的組態](#)
- StreamManagerClient 串流管理員 SDK 參考中的：
 - [Python](#)
 - [Java](#)
 - [Node.js](#)

匯出支援 AWS 雲端 目的地的組態

使用者定義的 Greengrass 元件會在 Stream Manager SDK StreamManagerClient 中使用 與串流管理員互動。當元件[建立串流](#)或[更新串流](#)時，它會傳遞代表串流屬性的 MessageStreamDefinition 物件，包括匯出定義。ExportDefinition 物件包含為串流定義的匯出組態。串流管理員使用這些匯出組態來判斷匯出串流的位置和方式。



您可以在串流上定義零或多個匯出組態，包括單一目的地類型的多個匯出組態。例如，您可以將串流匯出至兩個 AWS IoT Analytics 頻道和一個 Kinesis 資料串流。

對於失敗的匯出嘗試，串流管理員會持續重試 AWS 雲端 以最多五分鐘的間隔將資料匯出至 。重試嘗試次數沒有上限。

 Note

StreamManagerClient 也提供目標目的地，可用來將串流匯出至 HTTP 伺服器。此目標僅供測試之用。它不穩定或不支援在生產環境中使用。

支援的 AWS 雲端 目的地

- [AWS IoT Analytics 頻道](#)
- [Amazon Kinesis 資料串流](#)
- [AWS IoT SiteWise 資產屬性](#)
- [Amazon S3 物件](#)

您需負責維護這些 AWS 雲端 資源。

AWS IoT Analytics 頻道

串流管理員支援自動匯出至 AWS IoT Analytics。AWS IoT Analytics 可讓您對資料執行進階分析，以協助做出商業決策並改善機器學習模型。如需詳細資訊，請參閱AWS IoT Analytics 《使用者指南》中的[什麼是 AWS IoT Analytics ?](#)。

在 Stream Manager SDK 中，您的 Greengrass 元件會使用 IoTAnalyticsConfig 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [IoTAnalyticsConfig](#)
- Java 開發套件中的 [IoTAnalyticsConfig](#)
- Node.js SDK 中的 [IoTAnalyticsConfig](#)

要求

此匯出目的地有下列需求：

- 中的目標頻道 AWS IoT Analytics 必須與 Greengrass 核心裝置位於相同的 AWS 帳戶 和 AWS 區域中。
- 必須[授權核心裝置與 AWS 服務互動](#)允許 以頻道為目標的iotanalytics:BatchPutMessage許可。例如：

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"
      ]
    }
  ]
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的 [新增和移除 IAM 政策](#)。

匯出至 AWS IoT Analytics

若要建立匯出至的串流 AWS IoT Analytics，您的 Greengrass [元件會建立](#) 包含一或多個 IoTAnalyticsConfig 物件的匯出定義串流。此物件會定義匯出設定，例如目標頻道、批次大小、批次間隔和優先順序。

當您的 Greengrass 元件從裝置接收資料時，它們會將包含資料 Blob [的訊息附加](#) 到目標串流。

然後，串流管理員會根據串流匯出組態中定義的批次設定和優先順序匯出資料。

Amazon Kinesis 資料串流

串流管理員支援自動匯出至 Amazon Kinesis Data Streams。Kinesis Data Streams 通常用於彙總大量資料，並將其載入資料倉儲或 MapReduce 叢集。如需詳細資訊，請參閱 [《Amazon Kinesis 開發人員指南》](#) 中的 [什麼是 Amazon Kinesis Data Streams？](#)。Amazon Kinesis

在 Stream Manager SDK 中，您的 Greengrass 元件會使用 KinesisConfig 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [KinesisConfig](#)

- Java 開發套件中的 [KinesisConfig](#)
- Node.js SDK 中的 [KinesisConfig](#)

要求

此匯出目的地有下列需求：

- Kinesis Data Streams 中的目標串流必須與 Greengrass 核心裝置位於相同的 AWS 帳戶和 AWS 區域中。
- （建議）串流管理員 v2.2.1 可改善將串流匯出至 Kinesis Data Streams 目的地的效能。若要使用此最新版本的改進功能，請將[串流管理員元件](#)升級至 v2.2.1，並使用 [Greengrass 權杖交換角色](#)中的 `kinesis:ListShards` 政策。
- 必須[授權核心裝置與 AWS 服務互動](#)允許以資料串流為目標的 `kinesis:PutRecords` 許可。例如：

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"
      ]
    }
  ]
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》](#) 中的新增和移除 IAM 政策。

匯出至 Kinesis Data Streams

若要建立匯出至 Kinesis Data Streams 的串流，您的 Greengrass [元件會建立](#) 包含一或多個 KinesisConfig 物件的匯出定義串流。此物件會定義匯出設定，例如目標資料串流、批次大小、批次間隔和優先順序。

當您的 Greengrass 元件從裝置接收資料時，它們會將包含資料 Blob [的訊息附加](#) 到目標串流。然後，串流管理員會根據串流匯出組態中定義的批次設定和優先順序匯出資料。

串流管理員會為每個上傳至 Amazon Kinesis 的記錄產生唯一的隨機 UUID 做為分割區索引鍵。

AWS IoT SiteWise 資產屬性

串流管理員支援自動匯出至 AWS IoT SiteWise。AWS IoT SiteWise 可讓您大規模收集、組織和分析工業設備的資料。如需詳細資訊，請參閱 AWS IoT SiteWise 《使用者指南》中的 [什麼是 AWS IoT SiteWise ?](#)。

在 Stream Manager SDK 中，您的 Greengrass 元件會使用 IoTSiteWiseConfig 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [IoTSiteWiseConfig](#)
- Java 開發套件中的 [IoTSiteWiseConfig](#)
- Node.js SDK 中的 [IoTSiteWiseConfig](#)

Note

AWS 也提供 AWS IoT SiteWise 元件，提供預先建置的解決方案，可讓您用來從 OPC-UA 來源串流資料。如需詳細資訊，請參閱 [IoT SiteWise OPC UA 收集器](#)。

需求

此匯出目的地有下列需求：

- 中的目標資產屬性 AWS IoT SiteWise 必須與 Greengrass 核心裝置位於相同的 AWS 帳戶 和 AWS 區域 中。

Note

如需 AWS 區域 AWS IoT SiteWise 支援的清單，請參閱《AWS 一般參考》中的[AWS IoT SiteWise 端點和配額](#)。

- 必須[授權核心裝置與 AWS 服務互動](#)允許以資產屬性為目標的*iotsitewise:BatchPutAssetPropertyValue*許可。下列範例政策使用 *iotsitewise:assetHierarchyPath* 條件索引鍵來授予目標根資產及其子資產的存取權。您可以從 Condition 政策中移除，以允許存取所有 AWS IoT SiteWise 資產或指定個別資產 ARNs。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱《[IAM 使用者指南](#)》中的[新增和移除 IAM 政策](#)。

如需重要的安全性資訊，請參閱AWS IoT SiteWise 《使用者指南》中的[BatchPutAssetPropertyValue 授權](#)。

匯出至 AWS IoT SiteWise

若要建立匯出至的串流 AWS IoT SiteWise，您的 Greengrass [元件會建立](#) 包含一或多個 `IoTSiteWiseConfig` 物件的匯出定義串流。此物件會定義匯出設定，例如批次大小、批次間隔和優先順序。

當您的 Greengrass 元件從裝置接收資產屬性資料時，它們會將包含資料的訊息附加到目標串流。訊息是 JSON 序列化 `PutAssetPropertyValueEntry` 物件，其中包含一或多個資產屬性的屬性值。如需詳細資訊，請參閱 [附加訊息](#) 以取得 AWS IoT SiteWise 匯出目的地。

Note

當您傳送資料至 AWS IoT SiteWise，您的資料必須符合 `BatchPutAssetPropertyValue` 動作的要求。如需詳細資訊，請參閱《AWS IoT SiteWise API 參考》中的 [BatchPutAssetPropertyValue](#)。

然後，串流管理員會根據串流匯出組態中定義的批次設定和優先順序匯出資料。

您可以調整串流管理員設定和 Greengrass 元件邏輯，以設計匯出策略。例如：

- 對於近乎即時的匯出，請設定低批次大小和間隔設定，並在接收資料時將資料附加至串流。
- 為了最佳化批次處理、減輕頻寬限制或將成本降至最低，您的 Greengrass 元件可以在將資料附加至串流之前，將單一資產屬性收到的 `timestamp-quality-value (TQV)` 資料點合併。一種策略是在一個訊息中批次處理最多 10 個不同的屬性資產組合或屬性別名的項目，而不是為相同的屬性傳送多個項目。這有助於串流管理員保持在 [AWS IoT SiteWise 配額](#) 內。

Amazon S3 物件

串流管理員支援自動匯出至 Amazon S3。您可以使用 Amazon S3 來存放和擷取大量資料。如需詳細資訊，請參閱《[Amazon Simple Storage Service 開發人員指南](#)》中的 [什麼是 Amazon S3？](#)。

在 Stream Manager SDK 中，您的 Greengrass 元件會使用 `S3ExportTaskExecutorConfig` 來定義此目的地類型的匯出組態。如需詳細資訊，請參閱目標語言的 SDK 參考：

- Python SDK 中的 [S3ExportTaskExecutorConfig](#)
- Java 開發套件中的 [S3ExportTaskExecutorConfig](#)
- Node.js SDK 中的 [S3ExportTaskExecutorConfig](#)

要求

此匯出目的地有下列需求：

- 目標 Amazon S3 儲存貯體必須與 Greengrass 核心裝置 AWS 帳戶 位於相同的 中。
- 如果在 Greengrass 容器模式下執行的 Lambda 函數將輸入檔案寫入輸入檔案目錄，您必須將目錄掛載為具有寫入許可的容器中的磁碟區。這可確保檔案寫入根檔案系統，並在容器外部執行的串流管理員元件可見。
- 如果 Docker 容器元件將輸入檔案寫入輸入檔案目錄，您必須將目錄掛載為具有寫入許可的容器中的磁碟區。這可確保檔案寫入根檔案系統，並在容器外部執行的串流管理員元件可見。
- 必須[授權核心裝置與 AWS 服務互動](#)允許目標儲存貯體的下列許可。例如：

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/*"
      ]
    }
  ]
}
```

您可以使用萬用字元*命名機制，授予對資源的精細或條件式存取。如需詳細資訊，請參閱 [《IAM 使用者指南》中的新增和移除 IAM 政策](#)。

匯出至 Amazon S3

若要建立匯出至 Amazon S3 的串流，您的 Greengrass 元件會使用 S3ExportTaskExecutorConfig 物件來設定匯出政策。政策會定義匯出設定，例如分段上傳閾值和優先順序。對於 Amazon S3 匯出，串流管理員會上傳從核心裝置上的本機檔案讀取的資料。若要啟

動上傳，您的 Greengrass 元件會將匯出任務附加至目標串流。匯出任務包含輸入檔案和目標 Amazon S3 物件的相關資訊。串流管理員會以任務附加至串流的順序執行任務。

Note

目標儲存貯體必須已存在於您的 中 AWS 帳戶。如果指定金鑰的物件不存在，串流管理員會為您建立物件。

串流管理員使用分段上傳閾值屬性、[最小分段大小](#)設定和輸入檔案的大小來決定如何上傳資料。分段上傳閾值必須大於或等於最小分段大小。如果您想要平行上傳資料，您可以建立多個串流。

指定目標 Amazon S3 物件的金鑰可以在 `!{timestamp: value}` 預留位置中包含有效的 [Java DateTimeFormatter](#) 字串。您可以根據輸入檔案資料上傳的時間，使用這些時間戳記預留位置來分割 Amazon S3 中的資料。例如，下列金鑰名稱會解析為 等值 `my-key/2020/12/31/data.txt`。

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

Note

如果您想要監控串流的匯出狀態，請先建立狀態串流，然後將匯出串流設定為使用它。如需詳細資訊，請參閱[the section called “監控匯出任務”](#)。

管理輸入資料

您可以編寫程式碼，讓 IoT 應用程式用來管理輸入資料的生命週期。下列工作流程範例示範如何使用 Greengrass 元件來管理此資料。

1. 本機程序會從裝置或周邊接收資料，然後將資料寫入核心裝置上的目錄中的檔案。這些是串流管理員的輸入檔案。
2. Greengrass 元件會掃描目錄，並在[建立新檔案時將匯出任務附加](#)至目標串流。任務是 JSON 序列化 `S3ExportTaskDefinition` 物件，可指定輸入檔案的 URL、目標 Amazon S3 儲存貯體和金鑰，以及選用的使用者中繼資料。
3. 串流管理員會讀取輸入檔案，並依附加任務的順序將資料匯出至 Amazon S3。目標儲存貯體必須已存在於您的 中 AWS 帳戶。如果指定金鑰的物件不存在，串流管理員會為您建立物件。
4. Greengrass 元件會從狀態串流[讀取訊息](#)，以監控匯出狀態。匯出任務完成後，Greengrass 元件可以刪除對應的輸入檔案。如需詳細資訊，請參閱[the section called “監控匯出任務”](#)。

監控匯出任務

您可以編寫程式碼，讓 IoT 應用程式用來監控 Amazon S3 匯出的狀態。您的 Greengrass 元件必須建立狀態串流，然後將匯出串流設定為將狀態更新寫入狀態串流。單一狀態串流可以從匯出至 Amazon S3 的多個串流接收狀態更新。

首先，[建立](#)要用作狀態串流的串流。您可以設定串流的大小和保留政策，以控制狀態訊息的生命週期。例如：

- Memory 如果您不想儲存狀態訊息，請將 Persistence 設定為。
- StrategyOnFull 設定為，OverwriteOldestData 以免遺失新的狀態訊息。

然後，建立或更新匯出串流以使用狀態串流。具體而言，請設定串流 S3ExportTaskExecutorConfig 匯出組態的狀態組態屬性。此設定會告知串流管理員將有關匯出任務的狀態訊息寫入狀態串流。在 StatusConfig 物件中，指定狀態串流的名稱和詳細程度。下列支援的值範圍從最詳細 (ERROR) 到最詳細 (TRACE)。預設值為 INFO。

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

下列工作流程範例顯示 Greengrass 元件如何使用狀態串流來監控匯出狀態。

1. 如先前工作流程所述，Greengrass 元件[會將匯出任務附加](#)至設定為將有關匯出任務的狀態訊息寫入狀態串流的串流。附加操作會傳回代表任務 ID 的序號。
2. Greengrass 元件會從狀態串流循序[讀取訊息](#)，然後根據串流名稱和任務 ID 或根據訊息內容的匯出任務屬性來篩選訊息。例如，Greengrass 元件可以依匯出任務的輸入檔案 URL 進行篩選，該 URL 由訊息內容中的 S3ExportTaskDefinition 物件表示。

下列狀態碼表示匯出任務已達到已完成狀態：

- Success。上傳已成功完成。
- Failure。串流管理員遇到錯誤，例如，指定的儲存貯體不存在。解決問題後，您可以再次將匯出任務附加至串流。
- Canceled。任務已停止，因為串流或匯出定義已刪除，或任務的 time-to-live (TTL) 期間已過期。

 Note

任務也可能的狀態為 InProgress 或 Warning。當事件傳回不會影響任務執行的錯誤時，串流管理員會發出警告。例如，無法清除部分上傳會傳回警告。

3. 匯出任務完成後，Greengrass 元件可以刪除對應的輸入檔案。

下列範例顯示 Greengrass 元件如何讀取和處理狀態訊息。

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.

```

Python SDK 參考：[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.  
}
```

Java 開發套件參考：[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.
});
```

Node.js SDK 參考：[readMessages](#) | [StatusMessage](#)

執行機器學習推論

使用 AWS IoT Greengrass，您可以使用雲端訓練模型，在本機產生的資料上執行邊緣裝置上的機器學習 (ML) 推論。您可以從低延遲時間和節省執行本機推論成本中獲益，但仍同時充分利用雲端運算能力進行訓練模型和處理複雜地處理。

AWS IoT Greengrass 可讓執行推論所需的步驟更有效率。您可以在任何地方訓練推論模型，並將其部署為機器學習元件本機。例如，您可以在 [Amazon SageMaker AI](#) 中建置和訓練深度學習模型。然後，您可以將這些模型存放在 [Amazon S3](#) 儲存貯體中，以便將這些模型用作元件中的成品，在核心裝置上執行推論。

主題

- [AWS IoT Greengrass ML 推論的運作方式](#)
- [第 2 AWS IoT Greengrass 版有何不同？](#)
- [要求](#)
- [支援的模型來源](#)
- [支援的機器學習執行時間](#)
- [AWS提供的機器學習元件](#)
- [在 Greengrass 核心裝置上使用 Amazon SageMaker AI Edge Manager](#)
- [自訂您的機器學習元件](#)
- [故障診斷機器學習推論](#)

AWS IoT Greengrass ML 推論的運作方式

AWS 提供[機器學習元件](#)，可讓您用來建立單一步驟部署，以在裝置上執行機器學習推論。您也可以使用這些元件做為範本來建立自訂元件，以符合您的特定需求。

AWS 提供以下類別的機器學習元件：

- 模型元件 - 包含機器學習模型做為 Greengrass 成品。
- 執行期元件 - 包含在 Greengrass 核心裝置上安裝機器學習架構及其相依性的指令碼。
- 推論元件 - 包含推論程式碼，並包含元件相依性，以安裝機器學習架構並下載預先訓練的機器學習模型。

您為執行機器學習推論而建立的每個部署都包含至少一個執行推論應用程式的元件、安裝機器學習架構，以及下載機器學習模型。若要使用 AWS 提供的元件執行範例推論，您可以將推論元件部署到核心裝置，其會自動將對應的模型和執行期元件納入做為相依性。若要自訂部署，您可以使用自訂模型元件來插入或交換範例模型元件，也可以使用 AWS 所提供元件的元件配方做為範本，以建立您自己的自訂推論、模型和執行期元件。

若要使用自訂元件執行機器學習推論：

1. 建立模型元件。此元件包含您要用來執行 inference 的機器學習模型。AWS 提供範例預先訓練的 DLR 和 TensorFlow Lite 模型。若要使用自訂模型，請建立您自己的模型元件。
2. 建立執行期元件。此元件包含為模型安裝機器學習執行期所需的指令碼。AWS 提供[深度學習執行期 \(DLR\) 和 TensorFlow Lite 的範例執行期](#)元件。[TensorFlow](#) 若要將其他執行時間與自訂模型和推論程式碼搭配使用，請建立您自己的執行時間元件。
3. 建立推論元件。此元件包含您的推論程式碼，並將您的模型和執行期元件納入相依性。AWS 提供範例推論元件，以使用 DLR 和 TensorFlow Lite 進行影像分類和物件偵測。若要執行其他類型的推論，或使用自訂模型和執行時間，請建立您自己的推論元件。
4. 部署推論元件。當您部署此元件時，AWS IoT Greengrass 也會自動部署模型和執行期元件相依性。

若要開始使用 AWS 提供的元件，請參閱 [the section called “執行範例影像分類推論”](#)。

如需建立自訂機器學習元件的詳細資訊，請參閱 [自訂您的機器學習元件](#)。

第 2 AWS IoT Greengrass 版有何不同？

AWS IoT Greengrass 將機器學習的功能單位，例如模型、執行期和推論程式碼，合併成可讓您使用單一步驟程序來安裝機器學習執行期、下載訓練過的模型，以及在裝置上執行推論的元件。

透過使用 AWS 提供的機器學習元件，您可以靈活地使用範例推論程式碼和預先訓練的模型開始執行機器學習推論。您可以插入自訂模型元件，將您自己的自訂訓練模型與 AWS 提供的推論和執行期元件搭配使用。對於完全自訂的機器學習解決方案，您可以使用公有元件做為範本來建立自訂元件，並使用您想要的任何執行時間、模型或推論類型。

要求

若要建立和使用機器學習元件，您必須具備下列項目：

- Greengrass 核心裝置。如果您沒有帳戶，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。
- 至少 500 MB 的本機儲存空間，可使用 AWS 提供的範例機器學習元件。

支援的模型來源

AWS IoT Greengrass 支援使用儲存在 Amazon S3 中的自訂訓練機器學習模型。您也可以使用 Amazon SageMaker AI 邊緣封裝任務，直接為 SageMaker AI Neo 編譯模型建立模型元件。如需搭配使用 SageMaker AI Edge Manager 的詳細資訊 AWS IoT Greengrass，請參閱 [在 Greengrass 核心裝置上使用 Amazon SageMaker AI Edge Manager](#)。

包含模型的 S3 儲存貯體必須符合下列要求：

- 它們不得使用 SSE-C 加密。對於使用伺服器端加密的儲存貯體，AWS IoT Greengrass 機器學習推論目前僅支援 SSE-S3 或 SSE-KMS 加密選項。如需伺服器端加密選項的詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用伺服器端加密保護資料](#)。
- 其名稱不得包含句點 (.)。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的使用虛擬託管式儲存貯體搭配 SSL 的[儲存貯體命名規則](#)。
- 存放模型來源的 S3 儲存貯體必須與機器學習元件位於相同 AWS 帳戶和 AWS 區域。
- AWS IoT Greengrass 必須具有模型來源的 read 許可。若要讓 AWS IoT Greengrass 存取 S3 儲存貯體，[Greengrass 裝置角色](#)必須允許 s3:GetObject 動作。如需裝置角色的詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

支援的機器學習執行時間

AWS IoT Greengrass 可讓您建立自訂元件，以使用您選擇的任何機器學習執行時間，透過自訂訓練模型執行機器學習推論。如需建立自訂機器學習元件的詳細資訊，請參閱 [自訂您的機器學習元件](#)。

為了讓機器學習入門的過程更有效率，AWS IoT Greengrass 提供使用下列機器學習執行期的範例推論、模型和執行期元件：

- [深度學習執行期](#) (DLR) 1.6.0 版和 1.3.0 版
- [TensorFlow Lite](#) 2.5.0 版

AWS 提供的機器學習元件

下表列出用於機器學習的 AWS 所提供元件。

Note

數個 AWS 提供的元件取決於 Greengrass 核的特定次要版本。由於此相依性，當您將 Greengrass 核更新為新的次要版本時，您需要更新這些元件。如需每個元件所依賴之核的特定版本的資訊，請參閱對應的元件主題。如需更新核心的詳細資訊，請參閱 [更新 AWS IoT Greengrass 核心軟體 \(OTA\)](#)。

元件	說明	元件類型	支援的作業系統	開放原始碼
SageMaker AI Edge Manager	在 Greengrass 核心裝置上部署 Amazon SageMaker AI Edge Manager 代理程式。	一般	Linux、Windows	否
DLR 影像分類	使用 DLR 影像分類模型存放區和 DLR 執行期元件做為相依性的推論元件，以安裝 DLR、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。	一般	Linux、Windows	否
DLR 物件偵測	使用 DLR 物件偵測模型存放區和 DLR 執行期元件做為相依性的推論元件，以安裝	一般	Linux、Windows	否

元件	說明	元件類型	支援的作業系統	開放原始碼
	DLR、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。			
DLR 影像分類模型存放區	模型元件，其中包含做為 Greengrass 成品的範例 ResNet-50 影像分類模型。	一般	Linux、Windows	否
DLR 物件偵測模型存放區	模型元件，其中包含做為 Greengrass 成品的範例 YOLOv3 物件偵測模型。	一般	Linux、Windows	否
DLR 執行時間	包含安裝指令碼的執行期元件，用於在 Greengrass 核心裝置上安裝 DLR 及其相依性。	一般	Linux、Windows	否

元件	說明	元件類型	支援的作業系統	開放原始碼
TensorFlow Lite 影像分類	使用 TensorFlow Lite 影像分類模型存放區和 TensorFlow Lite 執行期元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例影像分類模型，以及在支援的裝置上執行影像分類推論。	一般	Linux、Windows	否
TensorFlow Lite 物件偵測	使用 TensorFlow Lite 物件偵測模型存放區和 TensorFlow Lite 執行期元件做為相依性的推論元件，以安裝 TensorFlow Lite、下載範例物件偵測模型，以及在支援的裝置上執行物件偵測推論。	一般	Linux、Windows	否

元件	說明	元件類型	支援的作業系統	開放原始碼
TensorFlow Lite 影像分類模型存放區	模型元件，其中包含做為 Greengrass 成品的範例 MobileNet v1 模型。	一般	Linux、Windows	否
TensorFlow Lite 物件偵測模型存放區	模型元件，其中包含做為 Greengrass 成品的範例單一鏡頭偵測 (SSD) MobileNet 模型。	一般	Linux、Windows	否
TensorFlow Lite 執行時間	包含安裝指令碼的執行期元件，用於在 Greengrass 核心裝置上安裝 TensorFlow Lite 及其相依性。	一般	Linux、Windows	否

在 Greengrass 核心裝置上使用 Amazon SageMaker AI Edge Manager

 Important

SageMaker AI Edge Manager 已於 2024 年 4 月 26 日終止。如需繼續將模型部署到邊緣裝置的詳細資訊，請參閱 [SageMaker AI Edge Manager 生命週期結束](#)。

Amazon SageMaker AI Edge Manager 是一種在邊緣裝置上執行的軟體代理程式。SageMaker AI Edge Manager 為邊緣裝置提供模型管理，讓您可以直接在 Greengrass 核心裝置上封裝和使用 Amazon SageMaker AI Neo 編譯的模型。透過使用 SageMaker AI Edge Manager，您也可以從核心裝置取樣模型輸入和輸出資料，並將該資料傳送至 AWS 雲端以進行監控和分析。由於 SageMaker AI Edge Manager 使用 SageMaker AI Neo 來最佳化目標硬體的模型，因此您不需要直接在裝置上安裝 DLR 執行期。在 Greengrass 裝置上，SageMaker AI Edge Manager 不會載入本機 AWS IoT 憑證或直接呼叫 AWS IoT 憑證提供者端點。反之，SageMaker AI Edge Manager 會使用[權杖交換服務](#)從 TES 端點擷取臨時憑證。

本節說明 SageMaker AI Edge Manager 如何在 Greengrass 核心裝置上運作。

SageMaker AI Edge Manager 如何在 Greengrass 裝置上運作

若要將 SageMaker AI Edge Manager 代理程式部署至您的核心裝置，請建立包含 `aws.greengrass.SageMakerEdgeManager component`。AWS IoT Greengrass manages 在裝置上安裝 Edge Manager 代理程式和生命週期的部署。當新版本的代理程式二進位檔可用時，請部署更新的 `aws.greengrass.SageMakerEdgeManager` 元件版本，以升級安裝在裝置上的代理程式版本。

當您搭配使用 SageMaker AI Edge Manager 時 AWS IoT Greengrass，您的工作流程包含下列高階步驟：

1. 使用 SageMaker AI Neo 編譯模型。
2. 使用 SageMaker AI 邊緣封裝任務封裝 SageMaker AI Neo 編譯模型。當您執行模型的邊緣封裝任務時，您可以選擇建立具有封裝模型的模型元件，做為可部署到 Greengrass 核心裝置的成品。
3. 建立自訂推論元件。您可以使用此推論元件與 Edge Manager 代理程式互動，以在核心裝置上執行推論。這些操作包括載入模型、叫用預測請求來執行推論，以及在元件關閉時卸載模型。
4. 部署 SageMaker AI Edge Manager 元件、封裝模型元件和推論元件，以在裝置上的 SageMaker AI 推論引擎 (Edge Manager 代理程式) 上執行模型。

如需有關建立與 SageMaker AI Edge Manager 搭配使用的邊緣封裝任務和推論元件的詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的[使用 部署模型套件和 Edge Manager Agent AWS IoT Greengrass](#)。

本[教學課程：開始使用 SageMaker AI Edge Manager](#)教學課程說明如何在現有的 Greengrass 核心裝置上設定和使用 SageMaker AI Edge Manager 代理程式，使用 AWS 您可用來建立範例推論和模型元件的範例程式碼。

當您在 Greengrass 核心裝置上使用 SageMaker AI Edge Manager 時，您也可以使用擷取資料功能將範例資料上傳至 AWS 雲端。擷取資料是您用來將推論輸入、推論結果和其他推論資料上傳至 S3 儲存貯體或本機目錄以供未來分析的 SageMaker AI 功能。如需搭配 SageMaker AI Edge Manager 使用擷取資料的詳細資訊，請參閱《Amazon SageMaker AI 開發人員指南》中的[管理模型](#)。

要求

您必須符合下列要求，才能在 Greengrass 核心裝置上使用 SageMaker AI Edge Manager 代理程式。

- 在 Amazon Linux 2 上執行的 Greengrass 核心裝置、以 Debian 為基礎的 Linux 平台 (x86_64 或 Armv8) 或 Windows (x86_64)。如果您沒有帳戶，請參閱 [教學課程：入門 AWS IoT Greengrass V2](#)。
- [Python](#) 3.6 或更新版本，包括 pip 安裝在核心裝置上的 Python 版本。
- 使用下列項目設定的 [Greengrass 裝置角色](#)：
 - 允許 `credentials.iot.amazonaws.com` 和 `sagemaker.amazonaws.com` 擔任角色的信任關係，如下列 IAM 政策範例所示。

```
{
  "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"
    }
  ]
}
```

- [AmazonSageMakerEdgeDeviceFleetPolicy](#) IAM 受管政策。
- `s3:PutObject` 動作，如下列 IAM 政策範例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- 在與 Greengrass 核心裝置相同的 AWS 帳戶 和 中建立 AWS 區域 的 Amazon S3 儲存貯體。SageMaker AI Edge Manager 需要 S3 儲存貯體來建立邊緣裝置機群，並儲存範例資料，避免在裝置上執行推論。如需建立 S3 儲存貯體的資訊，請參閱 [Amazon S3 入門](#)。
- 使用與 Greengrass 核心裝置相同 AWS IoT 角色別名的 SageMaker AI 邊緣裝置機群。如需詳細資訊，請參閱[建立邊緣裝置機群](#)。
- 您的 Greengrass 核心裝置已註冊為 SageMaker AI Edge 裝置機群中的邊緣裝置。邊緣裝置名稱必須與核心裝置的 AWS IoT 物件名稱相符。如需詳細資訊，請參閱[註冊您的 Greengrass 核心裝置](#)。

開始使用 SageMaker AI Edge Manager

您可以完成教學課程，開始使用 SageMaker AI Edge Manager。本教學課程說明如何開始使用 SageMaker AI Edge Manager AWS 搭配現有核心裝置上提供的範例元件。這些範例元件使用 SageMaker AI Edge Manager 元件做為相依性來部署 Edge Manager 代理程式，並使用使用 SageMaker AI Neo 編譯的預先訓練模型執行推論。如需詳細資訊，請參閱[教學課程：開始使用 SageMaker AI Edge Manager](#)。

自訂您的機器學習元件

在 中 AWS IoT Greengrass，您可以設定範例[機器學習元件](#)，以自訂如何在裝置上執行機器學習推論，並以推論、模型和執行期元件做為建置區塊。AWS IoT Greengrass 也可讓您靈活地使用範例元件做為範本，並視需要建立自己的自訂元件。您可以混合和比對此模組化方法，以下列方式自訂機器學習推論元件：

使用範例推論元件

- 當您部署推論元件時，請修改推論元件的組態。
- 使用自訂模型搭配範例推論元件，方法是將範例模型存放區元件取代為自訂模型元件。您的自訂模型必須使用與範例模型相同的執行時間進行訓練。

使用自訂推論元件

- 透過新增公有模型元件和執行期元件做為自訂推論元件的相依性，搭配範例模型和執行期使用自訂推論程式碼。
- 建立和新增自訂模型元件或執行期元件，做為自訂推論元件的相依性。如果您想要使用自訂推論程式碼，或未 AWS IoT Greengrass 提供範例元件的執行時間，則必須使用自訂元件。

主題

- [修改公有推論元件的組態](#)
- [搭配範例推論元件使用自訂模型](#)
- [建立自訂機器學習元件](#)
- [建立自訂推論元件](#)

修改公有推論元件的組態

在 [AWS IoT Greengrass 主控台](#) 中，元件頁面會顯示該元件的預設組態。例如，TensorFlow Lite 影像分類元件的預設組態如下所示：

```
{
  "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"
}
}
```

部署公有推論元件時，您可以修改預設組態來自訂部署。如需每個公有推論元件可用組態參數的相關資訊，請參閱 [中的元件主題AWS提供的機器學習元件](#)。

本節說明如何從 AWS IoT Greengrass 主控台部署修改過的元件。如需使用 部署元件的詳細資訊 AWS CLI，請參閱 [建立部署](#)。

部署修改過的公有推論元件（主控台）

1. 登入 [AWS IoT Greengrass 主控台](#)。
2. 在導覽功能表中，選擇元件。
3. 在元件頁面上的公有元件索引標籤上，選擇您要部署的元件。
4. 在元件頁面上，選擇部署。
5. 從新增至部署中，選擇下列其中一項：
 - a. 若要將此元件合併至目標裝置上的現有部署，請選擇新增至現有部署，然後選取要修訂的部署。
 - b. 若要在目標裝置上建立新部署，請選擇建立新部署。如果您的設備上有現有的部署，則選擇此步驟將取代現有部署。
6. 在指定目標頁面上，執行下列作業：
 - a. 在部署資訊下，輸入或修改部署的易記名稱。
 - b. 在部署目標下，選取部署的目標，然後選擇下一步。如果您要修訂既有部署，則無法變更部署目標。
7. 在選取元件頁面的公有元件下，驗證已選取具有已修改組態的推論元件，然後選擇下一步。
8. 在設定元件頁面上，執行下列動作：
 - a. 選取推論元件，然後選擇設定元件。
 - b. 在組態更新下，輸入您要更新的組態值。例如，在要合併的組態方塊中輸入下列組態更新，將推論間隔變更為 15 秒，並指示元件尋找/custom-ml-inference/images/資料夾中名為 custom.jpg 的映像。

```
{
  "InferenceInterval": "15",
  "ImageName": "custom.jpg",
  "ImageDirectory": "/custom-ml-inference/images/"
}
```

若要將元件的整個組態重設為其預設值，""請在重設路徑方塊中指定單一空白字串。

- c. 選擇確認，然後選擇下一步。
9. 在設定進階設定頁面上，保留預設組態設定，然後選擇下一步。
10. 在檢閱頁面上，選擇部署

搭配範例推論元件使用自訂模型

如果您想要將範例推論元件與您自己的機器學習模型搭配使用，以用於 AWS IoT Greengrass 提供範例執行期元件的執行期，您必須使用使用這些模型做為成品的元件覆寫公有模型元件。在高階，您需完成下列步驟，以搭配範例推論元件使用自訂模型：

1. 建立使用 S3 儲存貯體中自訂模型做為成品的模型元件。您的自訂模型必須使用與您要取代的模型相同的執行期進行訓練。
2. 修改推論元件中的ModelResourceKey組態參數，以使用自訂模型。如需更新推論元件組態的資訊，請參閱 [修改公有推論元件的組態](#)

當您部署推論元件時，AWS IoT Greengrass 尋找其元件相依性的最新版本。如果元件的較新自訂版本存在於相同的 AWS 帳戶 和 中，它會覆寫相依的公有模型元件 AWS 區域。

建立自訂模型元件（主控台）

1. 將模型上傳至 S3 儲存貯體。如需將模型上傳至 S3 儲存貯體的資訊，請參閱 [《Amazon Simple Storage Service 使用者指南》](#) 中的 [使用 Amazon S3 儲存貯體](#)。

Note

您必須將成品存放在與 AWS 帳戶 AWS 區域 元件相同的 S3 儲存貯體中。若要讓 AWS IoT Greengrass 存取這些成品，[Greengrass 裝置角色](#)必須允許 s3:GetObject動作。如需裝置角色的詳細資訊，請參閱 [授權核心裝置與 AWS 服務互動](#)。

2. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
3. 擷取公有模型存放區元件的元件配方。
 - a. 在元件頁面上的公有元件索引標籤上，尋找並選擇您要為其建立新版本的公有模型元件。例如 `variant.DLR.ImageClassification.ModelStore`。
 - b. 在元件頁面上，選擇檢視配方並複製顯示的 JSON 配方。
4. 在元件頁面上的我的元件索引標籤上，選擇建立元件。
5. 在建立元件頁面的元件資訊下，選取將配方輸入為 JSON 做為元件來源。
6. 在配方方塊中，貼上您先前複製的元件配方。
7. 在配方中，更新下列值：
 - `ComponentVersion`：增加元件的次要版本。

當您建立自訂元件以覆寫公有模型元件時，您只能更新現有元件版本的次要版本。例如，如果公有元件版本為 2.1.0，您可以使用版本 建立自訂元件 2.1.1。

- `Manifests.Artifacts.Uri`：將每個 URI 值更新為您要使用的模型的 Amazon S3 URI。

 Note

請勿變更元件的名稱。

8. 選擇建立元件。

建立自訂模型元件 (AWS CLI)

1. 將模型上傳至 S3 儲存貯體。如需將模型上傳至 S3 儲存貯體的資訊，請參閱《[Amazon Simple Storage Service 使用者指南](#)》中的[使用 Amazon S3 儲存貯體](#)。

 Note

您必須將成品存放在與 AWS 帳戶 AWS 區域 元件相同的 S3 儲存貯體中。若要讓 AWS IoT Greengrass 存取這些成品，[Greengrass 裝置角色](#)必須允許 `s3:GetObject` 動作。如需裝置角色的詳細資訊，請參閱 [授權核心裝置與 AWS 服務互動](#)。

2. 執行下列命令來擷取公有元件的元件配方。此命令會將元件配方寫入您在命令中提供的輸出檔案。視需要將擷取的 base64 編碼字串轉換為 JSON 或 YAML。

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. 將配方檔案名稱更新為 **<component-name>-<component-version>**，其中元件版本是新元件的目標版本。例如 `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`。
4. 在配方中，更新下列值：
 - **ComponentVersion**：增加元件的次要版本。

當您建立自訂元件以覆寫公有模型元件時，您只能更新現有元件版本的次要版本。例如，如果公有元件版本為 2.1.0，您可以使用版本 2.1.1 建立自訂元件。

- **Manifests.Artifacts.Uri**：將每個 URI 值更新為您要使用的模型的 Amazon S3 URI。

Note

請勿變更元件的名稱。

5. 執行下列命令，使用您擷取和修改的配方建立新的元件。

```
aws greengrassv2 create-component-version \  
  --inline-recipe fileb://path/to/component/recipe
```

Note

此步驟會在的服務中建立元件 AWS IoT Greengrass AWS 雲端。您可以使用 Greengrass CLI 在本機開發、測試和部署元件，然後再將其上傳至雲端。如需詳細資訊，請參閱[開發 AWS IoT Greengrass 元件](#)。

如需建立元件的詳細資訊，請參閱[開發 AWS IoT Greengrass 元件](#)。

建立自訂機器學習元件

如果您想要使用自訂推論程式碼，或未 AWS IoT Greengrass 提供範例元件的執行時間，則必須建立自訂元件。您可以使用自訂推論程式碼搭配 AWS 提供的範例機器學習模型和執行時間，也可以使用您自己的模型和執行時間來開發完全自訂的機器學習推論解決方案。如果您的模型使用 AWS IoT Greengrass 提供範例執行期元件的執行期，則您可以使用該執行期元件，而且您只需要為推論程式碼和您想要使用的模型建立自訂元件。

主題

- [擷取公有元件的配方](#)
- [擷取範例元件成品](#)
- [將元件成品上傳至 S3 儲存貯體](#)
- [建立自訂元件](#)

擷取公有元件的配方

您可以使用現有公有機器學習元件的配方做為範本來建立自訂元件。若要檢視公有元件最新版本的元件配方，請使用 主控台或 AWS CLI，如下所示：

- 使用主控台

1. 在元件頁面上的公有元件索引標籤上，尋找並選擇公有元件。
2. 在元件頁面上，選擇檢視配方。

- 使用 AWS CLI

執行下列命令來擷取公有變體元件的元件配方。此命令會將元件配方寫入您在命令中提供的 JSON 或 YAML 配方檔案。

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>
```

取代命令中的值，如下所示：

- **<arn>**。公有元件的 Amazon Resource Name (ARN)。
- **<recipe-format>**。您要建立配方檔案的格式。支援的值為 JSON 和 YAML。

- `<recipe-file>`。配方的名稱，格式為 `<component-name>-<component-version>`。

擷取範例元件成品

您可以使用公有機器學習元件所使用的成品做為範本，來建立自訂元件成品，例如推論程式碼或執行階段安裝指令碼。

若要檢視公有機器學習元件中包含的範例成品，請部署公有推論元件，然後在 `/greengrass/v2/packages/artifacts-unarchived/<component-name>/<component-version>/` 資料夾中檢視裝置上的成品。

將元件成品上傳至 S3 儲存貯體

您必須先將元件成品上傳至 S3 儲存貯體，並在元件配方中使用 S3 URIs，才能建立自訂元件。例如，若要在推論元件中使用自訂推論程式碼，請將程式碼上傳至 S3 儲存貯體。然後，您可以使用推論程式碼的 Amazon S3 URI 做為元件中的成品。

如需將內容上傳至 S3 儲存貯體的資訊，請參閱 [《Amazon Simple Storage Service 使用者指南》中的使用 Amazon S3 儲存貯體](#)。

Note

您必須將成品存放在與 AWS 帳戶 AWS 區域 元件相同的 S3 儲存貯體中。若要讓 AWS IoT Greengrass 存取這些成品，[Greengrass 裝置角色](#) 必須允許 `s3:GetObject` 動作。如需裝置角色的詳細資訊，請參閱 [授權核心裝置與 AWS 服務互動](#)。

建立自訂元件

您可以使用您擷取的成品和配方來建立自訂機器學習元件。如需範例，請參閱 [建立自訂推論元件](#)。

如需建立元件並將其部署至 Greengrass 裝置的詳細資訊，請參閱 [開發 AWS IoT Greengrass 元件](#) 和 [將 AWS IoT Greengrass 元件部署至裝置](#)。

建立自訂推論元件

本節說明如何使用 DLR 影像分類元件做為範本來建立自訂推論元件。

主題

- [將推論程式碼上傳至 Amazon S3 儲存貯體](#)
- [為您的推論元件建立配方](#)
- [建立推論元件](#)

將推論程式碼上傳至 Amazon S3 儲存貯體

建立推論程式碼，然後將其上傳至 S3 儲存貯體。如需將內容上傳至 S3 儲存貯體的資訊，請參閱《[Amazon Simple Storage Service 使用者指南](#)》中的[使用 Amazon S3 儲存貯體](#)。

Note

您必須將成品存放在與 AWS 帳戶 AWS 區域 元件相同的 S3 儲存貯體中。若要讓 AWS IoT Greengrass 存取這些成品，[Greengrass 裝置角色](#)必須允許 s3:GetObject 動作。如需裝置角色的詳細資訊，請參閱 [授權核心裝置與 AWS 服務互動](#)。

為您的推論元件建立配方

1. 執行下列命令來擷取 DLR 映像分類元件的元件配方。此命令會將元件配方寫入您在命令中提供的 JSON 或 YAML 配方檔案。

Linux, macOS, or Unix

```
aws greengrassv2 get-component \  
  --arn  
  arn:aws:greengrass:region:aws:components:aws.greengrass.DLRIImageClassification: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.DLRIImageClassification: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.DLRIImageClassification:versions
  `
  --recipe-output-format JSON | YAML `
  --query recipe `
  --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

將 *<recipe-file>* 取代為格式為 *<component-name>-<component-version>*。

- 在配方中的 ComponentDependencies 物件中，根據您要使用的模型和執行時間元件，執行下列一或多個動作：
 - 如果您想要使用 DLR 編譯模型，請保留 DLR 元件相依性。您也可以將其取代為自訂執行期元件的相依性，如下列範例所示。

執行期元件

JSON

```
{
  "<runtime-component>": {
    "VersionRequirement": "<version>",
    "DependencyType": "HARD"
  }
}
```

YAML

```
<runtime-component>:
  VersionRequirement: "<version>"
  DependencyType: HARD
```

- 保留 DLR 影像分類模型存放區相依性，以使用 AWS 提供的預先訓練 ResNet-50 模型，或修改它以使用自訂模型元件。當您包含公有模型元件的相依性時，如果元件的較新自訂版本存在於相同的中 AWS 帳戶 AWS 區域，則推論元件會使用該自訂元件。指定模型元件相依性，如下列範例所示。

公有模型元件

JSON

```
{
  "variant.DLR.ImageClassification.ModelStore": {
    "VersionRequirement": "<version>",
    "DependencyType": "HARD"
  }
}
```

YAML

```
variant.DLR.ImageClassification.ModelStore:
  VersionRequirement: "<version>"
  DependencyType: HARD
```

自訂模型元件

JSON

```
{
  "<custom-model-component>": {
    "VersionRequirement": "<version>",
    "DependencyType": "HARD"
  }
}
```

YAML

```
<custom-model-component>:
  VersionRequirement: "<version>"
  DependencyType: HARD
```

3. 在 ComponentConfiguration 物件中，新增此元件的預設組態。您稍後可以在部署元件時修改此組態。以下摘錄顯示 DLR 影像分類元件的元件組態。

例如，如果您使用自訂模型元件做為自訂推論元件的相依性，請修改 `ModelResourceKey` 以提供您正在使用的模型名稱。

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. 在 Manifests 物件中，提供元件部署到不同平台時所使用的成品和此元件組態的相關資訊，以及成功執行元件所需的任何其他資訊。以下摘錄顯示 DLR 映像分類元件中 Linux 平台的 Manifests 物件組態。

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
```

如需建立元件配方的詳細資訊，請參閱 [AWS IoT Greengrass 元件配方參考](#)。

建立推論元件

使用 AWS IoT Greengrass 主控台或 AWS CLI，使用您剛定義的配方來建立元件。建立元件之後，您可以部署它，在裝置上執行推論。如需如何部署推論元件的範例，請參閱 [教學課程：使用 TensorFlow Lite 執行範例影像分類推論](#)。

建立自訂推論元件（主控台）

1. 登入 [AWS IoT Greengrass 主控台](#)。
2. 在導覽功能表中，選擇元件。

3. 在元件頁面上的我的元件索引標籤上，選擇建立元件。
4. 在建立元件頁面的元件資訊下，選取將配方輸入為 JSON 或將配方輸入為 YAML 做為元件來源。
5. 在配方方塊中，輸入您建立的自訂配方。
6. 按一下建立元件。

建立自訂推論元件 (AWS CLI)

執行下列命令，使用您建立的配方建立新的自訂元件。

```
aws greengrassv2 create-component-version \  
  --inline-recipe fileb://path/to/recipe/file
```

Note

此步驟會在的服務中建立 AWS IoT Greengrass 元件 AWS 雲端。您可以使用 Greengrass CLI 在本機開發、測試和部署元件，然後再將其上傳至雲端。如需詳細資訊，請參閱[開發 AWS IoT Greengrass 元件](#)。

故障診斷機器學習推論

使用本節中的疑難排解資訊和解決方案，以協助解決機器學習元件的問題。如需公有機器學習推論元件，請參閱下列元件日誌中的錯誤訊息：

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`

如果元件已正確安裝，則元件日誌會包含用於推論的程式庫位置。

問題

- [無法擷取程式庫](#)
- [Cannot open shared object file](#)
- [Error: ModuleNotFoundError: No module named '<library>'](#)
- [未偵測到支援 CUDA 的裝置](#)
- [沒有此類檔案或目錄](#)
- [RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>](#)
- [picamera.exc.PiCameraError: Camera is not enabled](#)
- [記憶體錯誤](#)
- [磁碟空間錯誤](#)
- [逾時錯誤](#)

無法擷取程式庫

當安裝程式指令碼無法在 Raspberry Pi 裝置上部署期間下載所需的程式庫時，會發生下列錯誤。

```
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]
```

再次執行 `sudo apt-get update` 並部署您的元件。

Cannot open shared object file

當安裝程式指令碼無法在 Raspberry Pi 裝置上部署 `opencv-python` 期間下載所需的相依性時，您可能看到類似下列的錯誤。

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

執行下列命令來手動安裝的相依性 `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>'

當 ML 執行期程式庫或其相依性未正確安裝時，您可能會在 ML 執行期元件日誌 (`variant.DLR.log` 或 `variant.TensorFlowLite.log`) 中看到此錯誤。在下列情況下，可能會發生此錯誤：

- 如果您使用預設啟用的 `UseInstaller` 選項，則此錯誤表示 ML 執行期元件無法安裝執行期或其相依性。請執行下列操作：
 - 設定 ML 執行期元件以停用 `UseInstaller` 選項。
 - 安裝 ML 執行時間及其相依性，並將其提供給執行 ML 元件的系統使用者。如需詳細資訊，請參閱下列內容：
 - [DLR 執行時間 UseInstaller 選項](#)
 - [TensorFlow Lite 執行時間 UseInstaller 選項](#)
- 如果您不使用 `UseInstaller` 選項，此錯誤表示未為執行 ML 元件的系統使用者安裝 ML 執行時間或其相依性。請執行下列操作：
 - 檢查是否已為執行 ML 元件的系統使用者安裝程式庫。將 `ggc_user` 取代為系統使用者名稱，並將 `tflite_runtime` 取代為要檢查的程式庫名稱。

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\""
```

- 如果未安裝程式庫，請為該使用者安裝程式庫。將 `ggc_user` 取代為系統使用者的名稱，並將 `tflite_runtime` 取代為程式庫的名稱。

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 tf-lite-runtime"
```

如需每個 ML 執行時間相依性的詳細資訊，請參閱下列內容：

- [DLR 執行時間 UseInstaller 選項](#)
- [TensorFlow Lite 執行時間 UseInstaller 選項](#)

3. 如果問題仍然存在，請為其他使用者安裝程式庫，以確認此裝置是否可以安裝程式庫。例如，使用者可以是您的使用者、根使用者或管理員使用者。如果您無法為任何使用者成功安裝程式庫，您的裝置可能不支援程式庫。請參閱程式庫的文件，以檢閱需求並疑難排解安裝問題。

未偵測到支援 CUDA 的裝置

使用 GPU 加速時，您可能會看到下列錯誤。執行下列命令，為 Greengrass 使用者啟用 GPU 存取。

```
sudo usermod -a -G video ggc_user
```

沒有此類檔案或目錄

下列錯誤表示執行時間元件無法正確設定虛擬環境：

- *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

檢查日誌，以確保所有執行時間相依性都已正確安裝。如需安裝程式指令碼所安裝程式庫的詳細資訊，請參閱下列主題：

- [DLR 執行時間](#)
- [TensorFlow Lite 執行時間](#)

根據預設，*MLRootPath* 設定為 */greengrass/v2/work/component-name/greengrass_ml*。若要變更此位置，請直接在部署中包含 [DLR 執行時間](#) 或 [TensorFlow Lite 執行時間](#) 執行期元件，並在組態合併更新中指定 *MLRootPath* 參數的修改值。如需設定元件的詳細資訊，請參閱 [更新元件組態](#)。

Note

對於 DLR 元件 v1.3.x，您可以在推論元件的組態中設定 *MLRootPath* 參數，預設值為 *\$HOME/greengrass_ml*。

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

當您在執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 上執行機器學習推論時，您可能會看到下列錯誤。

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

發生此錯誤是因為 Raspberry Pi OS Bullseye 包含比 OpenCV 所需版本更舊的 NumPy 版本。若要修正此問題，請執行下列命令，將 NumPy 升級到最新版本。

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

當您在執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 上執行機器學習推論時，您可能會看到下列錯誤。

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
ensure that the camera has been enabled.
```

發生此錯誤是因為 Raspberry Pi OS Bullseye 包含與 ML 元件不相容的新攝影機堆疊。若要修正此問題，請啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

記憶體錯誤

當裝置沒有足夠的記憶體且元件程序中斷時，通常會發生下列錯誤。

- stderr. Killed.
- exitCode=137

我們建議至少部署 500 MB 的記憶體，以部署公有機器學習推論元件。

磁碟空間錯誤

no space left on device 錯誤通常發生在裝置沒有足夠的儲存空間時。在再次部署元件之前，請確定您的裝置上有足夠的可用磁碟空間。我們建議至少 500 MB 的可用磁碟空間，以部署公有機器學習推論元件。

逾時錯誤

公有機器學習元件會下載大於 200 MB 的大型機器學習模型檔案。如果下載在部署期間逾時，請檢查您的網際網路連線速度，然後重試部署。

使用 管理 Greengrass 核心裝置 AWS Systems Manager

Note

AWS IoT Greengrass 目前不支援 Windows 核心裝置上的此功能。

Systems Manager 是一種 AWS 服務，可用來檢視和控制您的基礎設施 AWS，包括 Amazon EC2 執行個體、內部部署伺服器 and 虛擬機器 VMs) 和邊緣裝置。Systems Manager 可讓您檢視操作資料、自動化操作任務，以及維護安全性和合規性。當您向 Systems Manager 註冊機器時，它稱為受管節點。如需詳細資訊，請參閱 AWS Systems Manager 《使用者指南》中的 [什麼是 AWS Systems Manager ?](#)。

Agent AWS Systems Manager (Systems Manager Agent) 是您可以在裝置上安裝的軟體，可讓 Systems Manager 更新、管理和設定它們。若要在 Greengrass 核心裝置上安裝 Systems Manager Agent，請部署 [Systems Manager Agent 元件](#)。當您第一次部署 Systems Manager Agent 時，它會將核心裝置註冊為 Systems Manager 受管節點。Systems Manager Agent 會在裝置上執行，以在中啟用與 Systems Manager 服務的通訊 AWS 雲端。如需如何安裝和設定 Systems Manager Agent 元件的詳細資訊，請參閱 [安裝 AWS Systems Manager 代理程式](#)。

Systems Manager 工具和功能稱為 功能。Greengrass 核心裝置支援所有 Systems Manager 功能。如需這些功能以及如何使用 Systems Manager 管理核心裝置的詳細資訊，請參閱 AWS Systems Manager 《使用者指南》中的 [Systems Manager 功能](#)。

AWS Systems Manager 為 Systems Manager 受管節點提供標準執行個體層和進階執行個體層。如果您是第一次使用 Systems Manager，請從 standard-instances 層開始。在 Standard-instances 層上，您可以在 AWS 區域中為每個註冊最多 1,000 個受管節點 AWS 帳戶。如果您需要在單一帳戶和區域中註冊超過 1,000 個受管節點，或者如果您需要使用 [Session Manager 功能](#)，請使用 advanced-instances 層。如需詳細資訊，請參閱 AWS Systems Manager 《使用者指南》中的 [設定執行個體層](#)。

主題

- [安裝 AWS Systems Manager 代理程式](#)
- [解除安裝 AWS Systems Manager 代理程式](#)

安裝 AWS Systems Manager 代理程式

Agent AWS Systems Manager (Systems Manager Agent) 是您安裝的 Amazon 軟體，可讓 Systems Manager 更新、管理和設定 Greengrass 核心裝置、Amazon EC2 執行個體和其他資源。代理程式會在中處理並執行 Systems Manager 服務的請求 AWS 雲端。然後，代理程式會將狀態和執行時間資訊傳回 Systems Manager 服務。如需詳細資訊，請參閱 AWS Systems Manager 《使用者指南》中的[關於 Systems Manager 代理](#)程式。

AWS 提供 Systems Manager Agent 做為 Greengrass 元件，您可以將其部署到您的 Greengrass 核心裝置，以使用 Systems Manager 管理它們。[Systems Manager Agent 元件](#)會安裝 Systems Manager Agent 軟體，並將核心裝置註冊為 Systems Manager 中的受管節點。遵循此頁面上的步驟來完成先決條件，並將 Systems Manager Agent 元件部署至核心裝置或核心裝置群組。

主題

- [步驟 1：完成一般 Systems Manager 設定步驟](#)
- [步驟 2：建立 Systems Manager 的 IAM 服務角色](#)
- [步驟 3：將許可新增至權杖交換角色](#)
- [步驟 4：部署 Systems Manager Agent 元件](#)
- [步驟 5：使用 Systems Manager 驗證核心裝置註冊](#)

步驟 1：完成一般 Systems Manager 設定步驟

如果您尚未這麼做，請完成的一般設定步驟 AWS Systems Manager。如需詳細資訊，請參閱 AWS Systems Manager 《使用者指南》中的[完成一般 Systems Manager 設定步驟](#)。

步驟 2：建立 Systems Manager 的 IAM 服務角色

Systems Manager Agent 使用 AWS Identity and Access Management (IAM) 服務角色與通訊 AWS Systems Manager。Systems Manager 會擔任此角色，在每個核心裝置上啟用 Systems Manager 功能。當您部署元件時，Systems Manager Agent 元件也會使用此角色將核心裝置註冊為 Systems Manager 受管節點。如果您尚未這麼做，請為 Systems Manager Agent 元件建立要使用的 Systems Manager 服務角色。如需詳細資訊，請參閱[《使用者指南》中的為邊緣裝置建立 IAM 服務角色](#)。

AWS Systems Manager

步驟 3：將許可新增至權杖交換角色

Greengrass 核心裝置使用稱為字符交換角色的 IAM 服務角色來與 AWS 服務互動。每個核心裝置都有您在[安裝 AWS IoT Greengrass 核心軟體](#)時建立的字符交換角色。許多 Greengrass 元件，例如 Systems Manager Agent，需要此角色的額外許可。Systems Manager 代理程式元件需要下列許可，其中包括使用您在 中建立之角色的許可[步驟 2：建立 Systems Manager 的 IAM 服務角色](#)。

```
{
  "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": "*"
    }
  ]
}
```

如果您尚未這麼做，請將這些許可新增至核心裝置的字符交換角色，以允許 Systems Manager Agent 操作。您可以將新政策新增至權杖交換角色，以授予此許可。

將許可新增至權杖交換角色（主控台）

1. 在 [IAM 主控台](#) 導覽功能表中，選擇角色。
2. 當您安裝 AWS IoT Greengrass Core 軟體時，請選擇您設定為字符交換角色的 IAM 角色。如果您在安裝 AWS IoT Greengrass Core 軟體時未指定權杖交換角色的名稱，則會建立名為 `GreengrassV2TokenExchangeRole` 的角色。
3. 在許可下，選擇新增許可，然後選擇連接政策。

4. 選擇建立政策。建立政策頁面會在新的瀏覽器索引標籤中開啟。
5. 在 Create policy (建立政策) 頁面上，執行下列動作：
 - a. 選擇 JSON 以開啟 JSON 編輯器。
 - b. 將以下 政策貼到 JSON 編輯器。以您在 中建立的服務角色名稱取代 *SSMServiceRole*[步驟 2：建立 Systems Manager 的 IAM 服務角色](#)。

```
{
  "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. 選擇下一步：標籤。
 - d. 選擇下一步：檢閱。
 - e. 為政策輸入 Name (名稱)，例如 **GreengrassSSMAgentComponentPolicy**。
 - f. 選擇建立政策。
 - g. 切換到您開啟權杖交換角色的上一個瀏覽器索引標籤。
6. 在新增許可頁面上，選擇重新整理按鈕，然後選取您在上一個步驟中建立的 Greengrass Systems Manager 代理程式政策。
7. 選擇連接政策。

使用此字符交換角色的核心裝置現在具有與 Systems Manager 服務互動的許可。

將許可新增至權杖交換角色 (AWS CLI)

新增授予 Systems Manager 使用許可的政策

1. 建立名為 `ssm-agent-component-policy.json` 的檔案，並將下列 JSON 複製到 檔案。以您在 `中` 建立的服務角色名稱取代 `SSMServiceRole`[步驟 2：建立 Systems Manager 的 IAM 服務角色](#)。

```
{
  "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. 執行下列命令，從 `中` 的政策文件建立政策 `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
```

從輸出中的政策中繼資料複製政策 Amazon Resource Name (ARN)。您可以在下一個步驟中使用此 ARN 將此政策連接至核心裝置角色。

3. 執行下列命令，將政策連接至字符交換角色。

- 以您在安裝 AWS IoT Greengrass Core 軟體時指定的字符交換角色名稱取代 *GreengrassV2TokenExchangeRole*。如果您在安裝 AWS IoT Greengrass Core 軟體時未指定權杖交換角色的名稱，則會建立名為的角色 *GreengrassV2TokenExchangeRole*。
- 將政策 ARN 取代為上一個步驟的 ARN。

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
```

如果命令沒有輸出，表示成功。使用此字符交換角色的核心裝置現在具有與 Systems Manager 服務互動的許可。

步驟 4：部署 Systems Manager Agent 元件

完成下列步驟以部署和設定 Systems Manager Agent 元件。您可以將元件部署至單一核心裝置或一組核心裝置。

部署 Systems Manager Agent 元件（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇元件。
2. 在元件頁面上，選擇公有元件索引標籤，然後選擇 `aws.greengrass.SystemsManagerAgent`。
3. 在 `aws.greengrass.SystemsManagerAgent` 頁面中，選擇部署。
4. 從新增至部署中，選擇要修改的現有部署，或選擇建立新的部署，然後選擇下一步。
5. 如果您選擇建立新的部署，請選擇部署的目標核心裝置或物件群組。在指定目標頁面的部署目標下，選擇核心裝置或物件群組，然後選擇下一步。
6. 在選取元件頁面上，確認已選取 `aws.greengrass.SystemsManagerAgent` 元件，選擇下一步。
7. 在設定元件頁面上，選取 `aws.greengrass.SystemsManagerAgent`，然後執行下列動作：
 - a. 選擇設定元件。
 - b. 在設定 `aws.greengrass.SystemsManagerAgent` 模態、在組態更新下，在要合併的組態中，輸入下列組態更新。以您在 中建立的服務角色名稱取代 `SSMServiceRole`[步驟 2：建立 Systems Manager 的 IAM 服務角色](#)。

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false
}
```

Note

如果核心裝置已執行以混合啟用註冊的 Systems Manager Agent，請將 `SSMOverrideExistingRegistration` 變更為 `true`。此參數指定 Systems Manager Agent 已在具有混合啟用的裝置上執行時，Systems Manager Agent 元件是否註冊核心裝置。

您也可以指定要新增至 Systems Manager Agent 元件為核心裝置建立之 Systems Manager 受管節點的標籤 (SSMResourceTags)。如需詳細資訊，請參閱 [Systems Manager Agent 元件組態](#)。

- c. 選擇確認以關閉模態，然後選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署可能需要一分鐘的時間才能完成。

部署 Systems Manager Agent 元件 (AWS CLI)

若要部署 Systems Manager Agent 元件，請建立包含在 components 物件 `aws.greengrass.SystemsManagerAgent` 中的部署文件，並指定元件的組態更新。遵循 中的指示 [建立部署](#) 建立新的部署或修改現有的部署。

下列範例部分部署文件指定 使用名為 的服務角色 `SSMServiceRole`。以您在 中建立的服務角色名稱取代 `SSMServiceRole` [步驟 2：建立 Systems Manager 的 IAM 服務角色](#)。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\": \"SSMServiceRole\",
        \"SSMOverrideExistingRegistration\": false}"
      }
    }
  }
}
```

Note

如果核心裝置已執行以混合啟用註冊的 Systems Manager Agent，請將 `SSMOverrideExistingRegistration` 變更為 `true`。此參數指定 Systems Manager Agent 已在具有混合啟用的裝置上執行時，Systems Manager Agent 元件是否註冊核心裝置。

您也可以指定要新增至 Systems Manager Agent 元件為核心裝置建立之 Systems Manager 受管節點的標籤 (SSMResourceTags)。如需詳細資訊，請參閱 [Systems Manager Agent 元件組態](#)。

可能需要幾分鐘才能完成部署。您可以使用 AWS IoT Greengrass 服務來檢查部署的狀態，也可以檢查 AWS IoT Greengrass Core 軟體日誌和 Systems Manager Agent 元件日誌，以確認 Systems Manager Agent 是否成功執行。如需詳細資訊，請參閱下列內容：

- [檢查部署狀態](#)
- [監控 AWS IoT Greengrass 日誌](#)
- AWS Systems Manager 《使用者指南》中的[檢視 Systems Manager Agent 日誌](#)

如果部署失敗或 Systems Manager Agent 未執行，您可以對每個核心裝置上的部署進行疑難排解。如需詳細資訊，請參閱下列內容：

- [故障診斷 AWS IoT Greengrass V2](#)
- AWS Systems Manager 《使用者指南》中的 [Systems Manager Agent 故障診斷](#)

步驟 5：使用 Systems Manager 驗證核心裝置註冊

Systems Manager Agent 元件執行時，會將核心裝置註冊為 Systems Manager 中的受管節點。您可以使用 AWS IoT Greengrass 主控台、Systems Manager 主控台和 Systems Manager API 來驗證核心裝置是否已註冊為受管節點。受管節點在主控台和 API AWS 的部分中也稱為執行個體。

驗證核心裝置註冊AWS IoT Greengrass（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇核心裝置。
2. 選擇要驗證的核心裝置。
3. 在核心裝置的詳細資訊頁面上，尋找AWS Systems Manager 執行個體屬性。如果此屬性存在並顯示 Systems Manager 主控台的連結，則核心裝置會註冊為受管節點。

您也可以找到 AWS Systems Manager ping 狀態屬性，以檢查核心裝置上的 Systems Manager 代理程式狀態。當狀態為線上時，您可以使用 Systems Manager 管理核心裝置。

驗證核心裝置註冊 (Systems Manager 主控台)

1. 在 [Systems Manager 主控台](#) 導覽功能表中，選擇機群管理員。
2. 在受管節點下，執行下列動作：
 - a. 新增來源類型為 的篩選條件AWS::IoT::Thing。
 - b. 新增篩選條件，其中來源 ID 是要驗證的核心裝置名稱。
3. 在受管節點資料表中尋找核心裝置。如果核心裝置位於 資料表中，則會將其註冊為受管節點。

您也可以找到 Systems Manager 代理程式 ping 狀態屬性，以檢查核心裝置上的 Systems Manager 代理程式狀態。當狀態為線上時，您可以使用 Systems Manager 管理核心裝置。

驗證核心裝置註冊 (AWS CLI)

- 使用 [DescribeInstanceInformation](#) 操作，取得符合您指定篩選條件的受管節點清單。執行下列命令來驗證核心裝置是否已註冊為受管節點。以要驗證的核心裝置名稱取代 *MyGreengrassCore*。

```
aws ssm describe-instance-information --filter  
Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
```

回應包含符合篩選條件的受管節點清單。如果清單包含受管節點，則核心裝置會註冊為受管節點。您也可以從回應中找到有關核心裝置受管節點的其他資訊。如果 PingStatus 屬性是 Online，您可以使用 Systems Manager 管理核心裝置。

在 Systems Manager 中確認核心裝置已註冊為受管節點後，您可以使用 Systems Manager 主控台和 API 來管理該核心裝置。如需可用來管理 Greengrass 核心裝置的 Systems Manager 功能的詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的 [Systems Manager 功能](#)。

解除安裝 AWS Systems Manager 代理程式

如果您不想再使用 管理 Greengrass 核心裝置 AWS Systems Manager，您可以從 Systems Manager 取消註冊核心裝置，並從裝置解除安裝 AWS Systems Manager Agent (Systems Manager Agent)。

您可以隨時重新註冊核心裝置。若要這樣做，請再次部署 Systems Manager Agent 元件，該元件會在安裝時向 Systems Manager 註冊核心裝置。Systems Manager 會將已取消註冊的核心裝置的命令歷史記錄存放 30 天。

主題

- [步驟 1：從 Systems Manager 取消註冊核心裝置](#)
- [步驟 2：解除安裝 Systems Manager Agent 元件](#)
- [步驟 3：解除安裝 Systems Manager Agent 軟體](#)

步驟 1：從 Systems Manager 取消註冊核心裝置

您可以使用 Systems Manager 主控台或 API 取消註冊核心裝置。如需詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的[取消註冊受管節點](#)。

步驟 2：解除安裝 Systems Manager Agent 元件

取消註冊核心裝置後，請從裝置解除安裝 [Systems Manager Agent 元件](#)。若要從 Greengrass 核心裝置移除元件，請修改安裝元件的部署，然後從部署中移除元件。AWS IoT Greengrass 核心軟體會在核心裝置的部署皆未指定該元件時解除安裝元件。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

解除安裝 Systems Manager Agent 元件（主控台）

1. 在[AWS IoT Greengrass 主控台](#)導覽功能表中，選擇核心裝置。
2. 選擇您要解除安裝 Systems Manager Agent 元件的核心裝置。
3. 在核心裝置詳細資訊頁面上，選擇部署索引標籤。
4. 選擇將 Systems Manager Agent 元件部署到核心裝置的部署。
5. 在部署詳細資訊頁面上，選擇修訂。
6. 在修訂部署模式中，選擇修訂部署。
7. 在步驟 1：指定目標中，選擇下一步。
8. 在步驟 2：選取元件中，清除aws.greengrass.SystemsManagerAgent元件的選擇，然後選擇下一步。
9. 在步驟 3：設定元件中，選擇下一步。
10. 在步驟 4：設定進階設定中，選擇下一步。
11. 在步驟 5：檢閱中，選擇部署。

解除安裝 Systems Manager Agent 元件 (CLI)

若要解除安裝 Systems Manager Agent 元件，請修改部署的部署，然後將其從部署中移除。如需詳細資訊，請參閱[修訂部署](#)。

可能需要幾分鐘才能完成部署。您可以使用 AWS IoT Greengrass 服務來檢查部署的狀態。如需詳細資訊，請參閱[檢查部署狀態](#)。

步驟 3：解除安裝 Systems Manager Agent 軟體

在您移除 Systems Manager Agent 元件後，Systems Manager Agent 軟體會繼續在核心裝置上執行。若要移除 Systems Manager Agent 軟體，您可以在核心裝置上執行命令。如需詳細資訊，請參閱AWS Systems Manager 《使用者指南》中的[從 Linux 執行個體解除安裝 Systems Manager Agent](#)。

中的安全性 AWS IoT Greengrass

的雲端安全性 AWS 是最高優先順序。身為 AWS 客戶，您可以受益於資料中心和網路架構，這些架構專為滿足最安全敏感組織的需求而建置。

安全性是 AWS 與您之間共同責任。[共同責任模式](#)將其描述為雲端的安全性，和雲端中的安全性：

- 雲端的安全性 – AWS 負責保護在 中執行 AWS 服務的基礎設施 AWS 雲端。AWS 也為您提供可安全使用的服務。在[AWS 合規計畫](#)中，第三方稽核人員會定期測試和驗證我們安全的有效性。若要了解適用的合規計劃 AWS IoT Greengrass，請參閱[AWS 合規計劃的服務範圍](#)。
- 雲端內部的安全 – 您的責任取決於所使用的 AWS 服務。您也必須對其他因素負責，包括資料的機密性、您公司的要求和適用法律和法規。

使用 時 AWS IoT Greengrass，您也必須負責保護裝置、本機網路連線和私有金鑰的安全。

本文件可協助您了解如何在使用 時套用共同責任模型 AWS IoT Greengrass。下列主題說明如何設定 AWS IoT Greengrass 以符合您的安全與合規目標。您也會了解如何使用其他 AWS 服務來協助您監控和保護 AWS IoT Greengrass 資源。

主題

- [中的資料保護 AWS IoT Greengrass](#)
- [的裝置身分驗證和授權 AWS IoT Greengrass](#)
- [的身分和存取管理 AWS IoT Greengrass](#)
- [允許透過代理或防火牆的裝置流量](#)
- [的合規驗證 AWS IoT Greengrass](#)
- [FIPS 端點](#)
- [中的彈性 AWS IoT Greengrass](#)
- [中的基礎設施安全性 AWS IoT Greengrass](#)
- [中的組態和漏洞分析 AWS IoT Greengrass](#)
- [中的程式碼完整性 AWS IoT Greengrass V2](#)
- [AWS IoT Greengrass 和介面 VPC 端點 \(AWS PrivateLink\)](#)
- [的安全最佳實務 AWS IoT Greengrass](#)

中的資料保護 AWS IoT Greengrass

AWS [共同責任模型](#)適用於 中的資料保護 AWS IoT Greengrass。如此模型所述，AWS 負責保護執行所有的全域基礎設施 AWS 雲端。您負責維護在此基礎設施上託管內容的控制權。您也同時負責所使用 AWS 服務的安全組態和管理任務。如需資料隱私權的詳細資訊，請參閱[資料隱私權常見問答集](#)。如需有關歐洲資料保護的相關資訊，請參閱AWS 安全性部落格上的[AWS 共同責任模型和 GDPR](#) 部落格文章。

基於資料保護目的，建議您保護 AWS 帳戶登入資料，並使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 設定個別使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用設定 API 和使用使用者活動記錄 AWS CloudTrail。如需有關使用 CloudTrail 追蹤擷取 AWS 活動的資訊，請參閱AWS CloudTrail 《使用者指南》中的[使用 CloudTrail 追蹤](#)。
- 使用 AWS 加密解決方案，以及其中的所有預設安全控制 AWS 服務。
- 使用進階的受管安全服務 (例如 Amazon Macie)，協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在 AWS 透過命令列界面或 API 存取時需要 FIPS 140-3 驗證的密碼編譯模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱[聯邦資訊處理標準 \(FIPS\) 140-3](#)。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如名稱欄位。這包括當您使用 AWS IoT Greengrass 或使用主控台、API AWS CLI或其他 AWS 服務 AWS SDKs 時。您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

如需在 中保護敏感資訊的詳細資訊 AWS IoT Greengrass，請參閱 [the section called “請勿記錄敏感資訊”](#)。

如需關於資料保護的詳細資訊，請參閱 AWS 安全部落格上的 [AWS 共同責任模型和歐盟《一般資料保護規範》\(GDPR\)](#) 部落格文章。

主題

- [資料加密](#)
- [硬體安全整合](#)

資料加密

AWS IoT Greengrass 使用加密來保護傳輸中（透過網際網路或本機網路）和靜態（存放在中）的資料 AWS 雲端。

AWS IoT Greengrass 環境中的裝置通常會收集傳送至 AWS 服務的資料，以供進一步處理。如需其他服務資料加密的詳細資訊 AWS，請參閱該服務的安全文件。

主題

- [傳輸中加密](#)
- [靜態加密](#)
- [Greengrass 核心裝置的金鑰管理](#)

傳輸中加密

AWS IoT Greengrass 有兩種通訊模式，其中資料正在傳輸中：

- [the section called “透過網際網路傳輸資料”](#)。Greengrass 核心與 AWS IoT Greengrass 網際網路之間的通訊會加密。
- [the section called “核心裝置上的資料”](#)。Greengrass 核心裝置上的元件之間的通訊不會加密。

透過網際網路傳輸資料

AWS IoT Greengrass 使用 Transport Layer Security (TLS) 來加密網際網路上的所有通訊。傳送到 的所有資料 AWS 雲端 都會使用 MQTT 或 HTTPS 通訊協定，透過 TLS 連線傳送，因此預設為安全。AWS IoT Greengrass 會使用 AWS IoT 傳輸安全模型。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[傳輸安全性](#)。

核心裝置上的資料

AWS IoT Greengrass 不會加密在 Greengrass 核心裝置上本機交換的資料，因為資料不會離開裝置。這包括使用者定義元件、AWS IoT 裝置 SDK 和公有元件之間的通訊，例如串流管理員。

靜態加密

AWS IoT Greengrass 會儲存您的資料：

- [the section called “中的靜態資料 AWS 雲端”](#)。此資料已加密。
- [the section called “Greengrass 核心上的靜態資料”](#)。此資料不會加密（您秘密的本機複本除外）。

中的靜態資料 AWS 雲端

AWS IoT Greengrass 會加密存放在 中的客戶資料 AWS 雲端。此資料使用 AWS KMS 管理的金鑰進行保護 AWS IoT Greengrass。

Greengrass 核心上的靜態資料

AWS IoT Greengrass 依賴 Unix 檔案許可和全磁碟加密（如果啟用）來保護核心上的靜態資料。保護檔案系統和裝置是您的責任。

不過，AWS IoT Greengrass 會加密擷取自 之秘密的本機複本 AWS Secrets Manager。如需詳細資訊，請參閱[秘密管理員](#)元件。

Greengrass 核心裝置的金鑰管理

客戶有責任保證在 Greengrass 核心裝置上安全儲存密碼編譯（公有和私有）金鑰。AWS IoT Greengrass 將公有和私有金鑰用於下列案例：

- IoT 用戶端金鑰會搭配 IoT 憑證在 Greengrass 核心連接至 AWS IoT Core 時，驗證 Transport Layer Security (TLS) 交握。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

Note

金鑰和憑證也稱為核心私有金鑰和核心裝置憑證。

Greengrass 核心裝置支援使用檔案系統許可或[硬體安全模組](#)的私有金鑰儲存。如果您使用以檔案系統為基礎的私有金鑰，您必須負責將其安全儲存於核心裝置上。

硬體安全整合

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

您可以將 AWS IoT Greengrass Core 軟體設定為透過 [PKCS#11 介面](#) 使用硬體安全模組 (HSM)。此功能可讓您安全地存放裝置的私有金鑰和憑證，使其不會在軟體中公開或複製。您可以將私有金鑰和憑證存放在硬體模組上，例如 HSM 或信任平台模組 (TPM)。

Core AWS IoT Greengrass 軟體使用私有金鑰和 X.509 憑證來驗證 AWS IoT 和 AWS IoT Greengrass 服務的連線。[秘密管理員元件](#)使用此私有金鑰安全地加密和解密您部署到 Greengrass 核心裝置的秘密。當您將核心裝置設定為使用 HSM 時，這些元件會使用您存放在 HSM 中的私有金鑰和憑證。

[Moquette MQTT 代理程式元件](#)也會為其本機 MQTT 伺服器憑證存放私有金鑰。此元件會將私有金鑰存放在元件工作資料夾中的裝置檔案系統上。目前，AWS IoT Greengrass 不支援將此私有金鑰或憑證儲存在 HSM 中。

Tip

在 [AWS 合作夥伴裝置目錄中搜尋支援此功能的裝置](#)。

主題

- [要求](#)
- [硬體安全最佳實務](#)
- [使用硬體安全性安裝 AWS IoT Greengrass 核心軟體](#)
- [在現有核心裝置上設定硬體安全性](#)
- [在沒有 PKCS#11 支援的情況下使用硬體](#)
- [另請參閱](#)

要求

您必須符合下列要求，才能在 Greengrass 核心裝置上使用 HSM：

- 核心裝置上已安裝 [Greengrass 核 v2.5.3](#) 或更新版本。您可以在核心裝置上安裝 AWS IoT Greengrass 核心軟體時，選擇相容的版本。
- 安裝在核心裝置上的 [PKCS#11 提供者元件](#)。您可以在核心裝置上安裝 AWS IoT Greengrass Core 軟體時下載並安裝此元件。
- 支援 [PKCS#1 v1.5](#) 簽章結構描述和具有 RSA-2048 金鑰大小（或更大）或 ECC 金鑰的 RSA 金鑰的硬體安全模組。

Note

若要搭配 ECC 金鑰使用硬體安全模組，您必須使用 [Greengrass nucleus v2.5.6](#) 或更新版本。

若要使用硬體安全模組和[秘密管理員](#)，您必須搭配 RSA 金鑰使用硬體安全模組。

- AWS IoT Greengrass 核心軟體可在執行時間（使用 libdl）載入的 PKCS#11 提供者程式庫，以叫用 PKCS#11 函數。PKCS#11 提供者程式庫必須實作下列 PKCS#11 API 操作：

- 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

- 硬體模組必須可透過插槽標籤根據 PKCS#11 規格中的定義來解析。

- 您必須將私有金鑰和憑證存放在相同插槽的 HSM 中，如果 HSM 支援物件 ID，則必須使用相同的物件標籤和物件 IDs。
- 憑證和私有金鑰必須透過物件標籤解析。
- 私有金鑰必須具有下列許可：
 - sign
 - decrypt
- (選用) 若要使用[秘密管理員元件](#)，您必須使用 2.1.0 版或更新版本，且私有金鑰必須具有下列許可：
 - unwrap
 - wrap

硬體安全最佳實務

在 Greengrass 核心裝置上設定硬體安全性時，請考慮下列最佳實務。

- 使用內部硬體隨機號碼產生器，直接在 HSM 產生私有金鑰。這種方法比匯入您在其他地方產生的私有金鑰更安全，因為私有金鑰會保留在 HSM 中。
- 將私有金鑰設定為不可變，並禁止匯出。
- 使用 HSM 硬體廠商建議的佈建工具，使用受硬體保護的私有金鑰產生憑證簽署請求 (CSR)，然後使用 AWS IoT 主控台或 API 產生用戶端憑證。

Note

當您在 HSM 上產生私有金鑰時，輪換金鑰的安全最佳實務不適用。

使用硬體安全性安裝 AWS IoT Greengrass 核心軟體

安裝 AWS IoT Greengrass Core 軟體時，您可以將其設定為使用您在 HSM 中產生的私有金鑰。此方法遵循[安全最佳實務](#)，以在 HSM 中產生私有金鑰，因此私有金鑰會保留在 HSM 中。

若要安裝具有硬體安全性 AWS IoT Greengrass 的核心軟體，請執行下列動作：

1. 在 HSM 中產生私有金鑰。
2. 從私有金鑰建立憑證簽署請求 (CSR)。

3. 從 CSR 建立憑證。您可以建立由 AWS IoT 或其他根憑證授權機構 (CA) 簽署的憑證。如需如何使用另一個根 CA 的詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[建立您自己的用戶端憑證](#)。
4. 下載 AWS IoT 憑證並將其匯入 HSM。
5. 從組態檔案安裝 AWS IoT Greengrass Core 軟體，指定使用 PKCS#11 提供者元件，以及 HSM 中的私有金鑰和憑證。

您可以選擇下列其中一個安裝選項，以安裝具有硬體安全性的 AWS IoT Greengrass Core 軟體：

- 手動安裝

選擇此選項可手動建立所需的 AWS 資源並設定硬體安全性。如需詳細資訊，請參閱[使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

- 使用自訂佈建進行安裝

選擇此選項可開發自訂 Java 應用程式，以自動建立所需的 AWS 資源並設定硬體安全性。如需詳細資訊，請參閱[使用自訂資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

目前，當您使用[自動資源佈建](#)或[AWS IoT 機群佈建](#)安裝時，AWS IoT Greengrass 不支援安裝具有硬體安全性 AWS IoT Greengrass 的核心軟體。

在現有核心裝置上設定硬體安全性

您可以將核心裝置的私有金鑰和憑證匯入 HSM，以設定硬體安全性。

考量事項

- 您必須擁有核心裝置檔案系統的根存取權。
- 在此程序中，您會關閉 AWS IoT Greengrass Core 軟體，因此核心裝置會在您設定硬體安全性時離線且無法使用。

若要在現有核心裝置上設定硬體安全性，請執行下列動作：

1. 初始化 HSM。
2. 將 [PKCS#11 提供者元件](#) 部署至核心裝置。
3. 停止 AWS IoT Greengrass 核心軟體。

4. 將核心裝置的私有金鑰和憑證匯入 HSM。
5. 更新 AWS IoT Greengrass Core 軟體的組態檔案，以在 HSM 中使用私有金鑰和憑證。
6. 啟動 AWS IoT Greengrass 核心軟體。

步驟 1：初始化硬體安全模組

完成下列步驟，以初始化核心裝置上的 HSM。

初始化硬體安全模組

- 在 HSM 中初始化 PKCS#11 字符，並儲存該字符的槽 ID 和使用者 PIN。檢查 HSM 的文件，了解如何初始化權杖。您稍後在部署和設定 PKCS#11 提供者元件時，會使用插槽 ID 和使用者 PIN 碼。

步驟 2：部署 PKCS#11 提供者元件

請完成下列步驟，以部署和設定 [PKCS#11 提供者元件](#)。您可以將元件部署到一或多個核心裝置。

部署 PKCS#11 提供者元件（主控台）

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇元件。
2. 在元件頁面上，選擇公有元件索引標籤，然後選擇 `aws.greengrass.crypto.Pkcs11Provider`。
3. 在 `aws.greengrass.crypto.Pkcs11Provider` 頁面中，選擇部署。
4. 從新增至部署中，選擇要修改的現有部署，或選擇建立新的部署，然後選擇下一步。
5. 如果您選擇建立新的部署，請選擇部署的目標核心裝置或物件群組。在指定目標頁面的部署目標下，選擇核心裝置或物件群組，然後選擇下一步。
6. 在選取元件頁面的公有元件下，選取 `aws.greengrass.crypto.Pkcs11Provider`，然後選擇下一步。
7. 在設定元件頁面上，選取 `aws.greengrass.crypto.Pkcs11Provider`，然後執行下列動作：
 - a. 選擇設定元件。
 - b. 在設定 `aws.greengrass.crypto.Pkcs11Provider` 模態、在組態更新下，在要合併的組態中，輸入下列組態更新。使用目標核心裝置的值更新下列組態參數。指定您先前初始化 PKCS#11 字符的槽 ID 和使用者 PIN 碼。您稍後會將私有金鑰和憑證匯入 HSM 中的此插槽。

name

PKCS#11 組態的名稱。

library

AWS IoT Greengrass 核心軟體可以使用 libdl 載入之 PKCS#11 實作程式庫的絕對檔案路徑。

slot

包含私有金鑰和裝置憑證的插槽 ID。此值與槽索引或槽標籤不同。

userPin

用來存取插槽的使用者 PIN 碼。

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

- c. 選擇確認以關閉模態，然後選擇下一步。
8. 在設定進階設定頁面上，保留預設組態設定，然後選擇 下一步。
9. 在 Review (檢閱) 頁面，選擇 Deploy (部署)。

部署可能需要一分鐘的時間才能完成。

部署 PKCS#11 提供者元件 (AWS CLI)

若要部署 PKCS#11 提供者元件，請建立包含在 components 物件 `aws.greengrass.crypto.Pkcs11Provider` 中的部署文件，並指定元件的組態更新。遵循 [中的指示](#) [建立部署](#) 建立新的部署或修改現有的部署。

下列範例部分部署文件會指定 部署和設定 PKCS#11 提供者元件。使用目標核心裝置的值更新下列組態參數。儲存插槽 ID 和使用者的 PIN 碼，以供稍後將私有金鑰和憑證匯入 HSM 時使用。

name

PKCS#11 組態的名稱。

library

AWS IoT Greengrass 核心軟體可以使用 libdl 載入之 PKCS#11 實作程式庫的絕對檔案路徑。

slot

包含私有金鑰和裝置憑證的插槽 ID。此值與槽索引或槽標籤不同。

userPin

用來存取插槽的使用者 PIN 碼。

```
{
  "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\"}"
      }
    }
  }
}
```

可能需要幾分鐘才能完成部署。您可以使用 AWS IoT Greengrass 服務來檢查部署的狀態。您可以檢查 AWS IoT Greengrass 核心軟體日誌，以確認 PKCS#11 供應商元件已成功部署。如需詳細資訊，請參閱下列內容：

- [檢查部署狀態](#)
- [監控 AWS IoT Greengrass 日誌](#)

如果部署失敗，您可以對每個核心裝置上的部署進行疑難排解。如需詳細資訊，請參閱[故障診斷 AWS IoT Greengrass V2](#)。

步驟 3：更新核心裝置上的組態

Core AWS IoT Greengrass 軟體使用指定裝置運作方式的組態檔案。此組態檔案包含在哪裡尋找裝置用來連線至的私有金鑰和憑證 AWS 雲端。完成下列步驟，將核心裝置的私有金鑰和憑證匯入 HSM，並更新組態檔案以使用 HSM。

更新核心裝置上的組態以使用硬體安全性

1. 停止 AWS IoT Greengrass 核心軟體。如果您將 [AWS IoT Greengrass Core 軟體設定為使用 systemd 的系統服務](#)，您可以執行下列命令來停止軟體。

```
sudo systemctl stop greengrass.service
```

2. 尋找核心裝置的私有金鑰和憑證檔案。

- 如果您使用 [自動佈建](#) 或 [機群佈建](#) 安裝 AWS IoT Greengrass 核心軟體，私有金鑰存在於 `/greengrass/v2/privKey.key`，而憑證存在於 `/greengrass/v2/thingCert.crt`。
- 如果您使用 [手動佈建](#) 安裝 AWS IoT Greengrass 核心軟體，`/greengrass/v2/private.pem.key` 則預設情況下私有金鑰位於，且憑證 `/greengrass/v2/device.pem.crt` 預設存在於。

您也可以在中檢查 `system.privateKeyPath` 和 `system.certificateFilePath` 屬性 `/greengrass/v2/config/effectiveConfig.yaml`，以尋找這些檔案的位置。

3. 將私有金鑰和憑證匯入 HSM。檢查 HSM 的文件，了解如何將私有金鑰和憑證匯入其中。使用您先前初始化 PKCS#11 字符的槽 ID 和使用者 PIN 匯入私有金鑰和憑證。您必須為私有金鑰和憑證使用相同的物件標籤和物件 ID。儲存您在匯入每個檔案時指定的物件標籤。您稍後更新 AWS IoT Greengrass Core 軟體組態以使用 HSM 中的私有金鑰和憑證時，會使用此標籤。
4. 更新 AWS IoT Greengrass 核心組態以使用 HSM 中的私有金鑰和憑證。若要更新組態，請修改 AWS IoT Greengrass Core 組態檔案，並使用更新的組態檔案執行 AWS IoT Greengrass Core 軟體，以套用新的組態。

請執行下列操作：

- a. 建立 AWS IoT Greengrass Core 組態檔案的備份。如果您在設定硬體安全性時遇到問題，您可以使用此備份來還原核心裝置。

```
sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
```

- b. 在文字編輯器中開啟 AWS IoT Greengrass 核心組態檔案。例如，您可以執行下列命令來使用 GNU nano 來編輯檔案。`/greengrass/v2` 將取代為 Greengrass 根資料夾的路徑。

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

- c. 將 `system.privateKeyPath` 值取代為 HSM 中私有金鑰的 PKCS#11 URI。將 `iotdevicekey` 取代為您先前匯入私有金鑰和憑證的物件標籤。

```
pkcs11:object=iotdevicekey;type=private
```

- d. 將 `system.certificateFilePath` 值取代為 HSM 中憑證的 PKCS#11 URI。將 `iotdevicekey` 取代為您先前匯入私有金鑰和憑證的物件標籤。

```
pkcs11:object=iotdevicekey;type=cert
```

完成這些步驟後，AWS IoT Greengrass 核心組態檔案中的 `system` 屬性看起來應該類似下列範例。

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. 在更新的 `effectiveConfig.yaml` 檔案中套用組態。Greengrass.jar 使用 `--init-config` 參數執行，以在 `effectiveConfig.yaml` 中套用組態。`/greengrass/v2` 將取代為 Greengrass 根資料夾的路徑。

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. 啟動 AWS IoT Greengrass 核心軟體。如果您將 [AWS IoT Greengrass Core 軟體](#) 設定為具有 [systemd 的系統服務](#)，您可以執行下列命令來啟動軟體。

```
sudo systemctl start greengrass.service
```

如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)。

7. 檢查 AWS IoT Greengrass 核心軟體日誌，以確認軟體啟動並連線至 AWS 雲端。Core AWS IoT Greengrass 軟體使用私有金鑰和憑證來連線至 AWS IoT 和 AWS IoT Greengrass 服務。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

下列 INFO 層級日誌訊息表示 AWS IoT Greengrass Core 軟體已成功連線至 AWS IoT 和 AWS IoT Greengrass 服務。

```
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. (選用) 驗證 AWS IoT Greengrass Core 軟體是否可與 HSM 中的私有金鑰和憑證搭配使用後，請從裝置的檔案系統刪除私有金鑰和憑證檔案。執行下列命令，並將檔案路徑取代為私有金鑰和憑證檔案的路徑。

```
sudo rm /greengrass/v2/privKey.key
sudo rm /greengrass/v2/thingCert.crt
```

在沒有 PKCS#11 支援的情況下使用硬體

PKCS#11 程式庫通常是硬體廠商提供，或是開放原始碼。例如，使用標準相容硬體 (例如 TPM1.2) 時，可以使用現有的開放原始碼軟體。不過，如果您的硬體沒有對應的 PKCS#11 程式庫實作，或者如果您想要撰寫自訂 PKCS#11 供應商，請聯絡您的 Amazon Web Services Enterprise Support 代表，詢問整合相關問題。

另請參閱

- [PKCS #11 密碼編譯字符界面使用者指南 2.4.0 版](#)
- [RFC 7512](#)
- [PKCS #1: RSA Encryption Version 1.5](#)

的裝置身分驗證和授權 AWS IoT Greengrass

AWS IoT Greengrass 環境中的裝置使用 X.509 憑證進行身分驗證，並使用 AWS IoT 政策進行授權。憑證和原則允許裝置彼此、與 AWS IoT Core 和 AWS IoT Greengrass 安全地連線。

X.509 憑證為數位憑證，採用 X.509 公有金鑰基礎架構標準，將公有金鑰與憑證內含的身分建立關聯。X.509 憑證是由稱為憑證授權機構 (CA) 的受信任實體所發行。CA 負責維護一個或多個稱為憑證授權機構憑證的特殊憑證，用以發行 X.509 憑證。僅憑證授權機構可存取憑證授權機構憑證。

AWS IoT 政策定義 AWS IoT 裝置允許的一組操作。具體而言，它們允許和拒絕對 AWS IoT Core 和 AWS IoT Greengrass 資料平面操作的存取，例如發佈 MQTT 訊息和擷取裝置影子。

所有裝置都需要 AWS IoT Core 登錄檔中的項目，以及已連接 AWS IoT 政策的已啟用 X.509 憑證。裝置可分為兩類：

- Greengrass 核心裝置

Greengrass 核心裝置使用憑證和 AWS IoT 政策來連線至 AWS IoT Core 和 AWS IoT Greengrass。憑證和政策也允許 AWS IoT Greengrass 將元件和組態部署到核心裝置。

- 用戶端裝置

MQTT 用戶端裝置使用憑證和政策來連線至 AWS IoT Core AWS IoT Greengrass 和服務。這可讓用戶端裝置使用 AWS IoT Greengrass 雲端探索來尋找和連線至 Greengrass 核心裝置。用戶端裝置使用相同的憑證來 AWS IoT Core 連線至雲端服務和核心裝置。用戶端裝置也會使用探索資訊與核心裝置進行交互身分驗證。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

X.509 憑證

必須 AWS IoT Greengrass 驗證核心裝置和用戶端裝置之間的通訊，以及裝置和 AWS IoT Core 或之間的通訊。此相互驗證是根據已註冊的 X.509 裝置憑證和加密金鑰。

在 AWS IoT Greengrass 環境中，裝置使用具有公有和私有金鑰的憑證進行下列 Transport Layer Security (TLS) 連線：

- Greengrass 核心裝置上的 AWS IoT 用戶端元件，可透過網際網路連線至 AWS IoT Core 和 AWS IoT Greengrass。
- AWS IoT Greengrass 透過網際網路連線至 的用戶端裝置，以探索核心裝置。
- Greengrass 核心上的 MQTT 代理程式元件會透過本機網路連接到群組中的 Greengrass 裝置。

AWS IoT Greengrass 核心裝置會將憑證存放在 Greengrass 根資料夾中。

憑證授權機構 (CA) 憑證

Greengrass 核心裝置和用戶端裝置會下載根 CA 憑證，用於使用 AWS IoT Core 和 AWS IoT Greengrass 服務進行身分驗證。我們建議您使用 Amazon Trust Service (ATS) 根憑證授權機構憑證，例如 [Amazon 根 CA 1](#)。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[伺服器身分驗證的 CA 憑證](#)。

用戶端裝置也會下載 Greengrass 核心裝置 CA 憑證。在相互身分驗證期間，他們會使用此憑證來驗證核心裝置上的 MQTT 伺服器憑證。

本機 MQTT 代理程式上的憑證輪換

當您[啟用用戶端裝置支援](#)時，Greengrass 核心裝置會產生本機 MQTT 伺服器憑證，供用戶端裝置用於交互身分驗證。此憑證由核心裝置 CA 憑證簽署，核心裝置會存放在 AWS IoT Greengrass 雲端。用戶端裝置會在發現核心裝置時擷取核心裝置 CA 憑證。當連接至核心裝置時，他們會使用核心裝置 CA 憑證來驗證核心裝置的 MQTT 伺服器憑證。核心裝置 CA 憑證會在 5 年後過期。

MQTT 伺服器憑證預設會每 7 天過期一次，而且您可以將此持續時間設定為 2 到 10 天。此有限期間是依據安全最佳實務。此輪換有助於減輕攻擊者竊取 MQTT 伺服器憑證和私有金鑰以模擬 Greengrass 核心裝置的威脅。

Greengrass 核心裝置會在 MQTT 伺服器憑證過期前 24 小時輪換 MQTT 伺服器憑證。Greengrass 核心裝置會產生新的憑證，並重新啟動本機 MQTT 代理程式。發生這種情況時，連接至 Greengrass 核心裝置的所有用戶端裝置都會中斷連線。用戶端裝置可以在短時間內重新連線至 Greengrass 核心裝置。

AWS IoT 資料平面操作的 政策

使用 AWS IoT 政策來授權存取 AWS IoT Core 和 AWS IoT Greengrass 資料平面。AWS IoT Core 資料平面可為裝置、使用者和應用程式提供操作。這些操作包括連線至 AWS IoT Core 和訂閱主題的功能。AWS IoT Greengrass 資料平面提供 Greengrass 裝置的操作。如需詳細資訊，請參閱[AWS IoT Greengrass V2 政策動作](#)。這些操作包括解析元件相依性和下載公有元件成品的能力。

AWS IoT 政策是類似 [IAM 政策](#) 的 JSON 文件。它包含一或多個指定下列屬性的政策陳述式：

- **Effect。** 存取模式，可以是 Allow 或 Deny。
- **Action。** 政策允許或拒絕的動作清單。
- **Resource。** 允許或拒絕動作的資源清單。

AWS IoT 政策支援*萬用字元，並將 MQTT 萬用字元 (+ 和 #) 視為文字字串。如需*萬用字元的詳細資訊，請參閱AWS Identity and Access Management 《使用者指南》[中的在資源 ARNs 中使用萬用字元](#)。

如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[AWS IoT 政策和AWS IoT 政策動作](#)。

Important

核心裝置或 Greengrass 資料平面操作的政策中不支援的[物件政策變數](#) (iot:Connection.Thing.*)。AWS IoT 反之，您可以使用與具有類似名稱的多個裝置相符的萬用字元。例如，您可以指定 MyGreengrassDevice*來比對 MyGreengrassDevice1、MyGreengrassDevice2等。

Note

AWS IoT Core 可讓您將 AWS IoT 政策連接至物件群組，以定義裝置群組的許可。物件群組政策不允許存取 AWS IoT Greengrass 資料平面操作。若要允許物件存取 AWS IoT Greengrass 資料平面操作，請將許可新增至您連接至物件憑證 AWS IoT 的政策。

AWS IoT Greengrass V2 政策動作

AWS IoT Greengrass V2 定義下列政策動作，Greengrass 核心裝置和用戶端裝置可在 AWS IoT 政策中使用。若要指定政策動作的資源，您可以使用資源的 Amazon Resource Name (ARN)。

核心裝置動作

`greengrass:GetComponentVersionArtifact`

准許取得預先簽章的 URL，以下載公有元件成品或 Lambda 元件成品。

當核心裝置收到指定公有元件或具有成品之 Lambda 的部署時，會評估此許可。如果核心裝置已有成品，則不會再次下載成品。

資源類型：componentVersion

資源 ARN 格式：arn:aws:greengrass:*region*:*account-id*:components:*component-name*:versions:*component-version*

greengrass:ResolveComponentCandidates

准許識別符合部署元件、版本和平台需求的元件清單。如果需求衝突，或不存在符合需求的元件，此操作會傳回錯誤，且裝置上的部署會失敗。

當核心裝置收到指定元件的部署時，會評估此許可。

資源類型：無

資源 ARN 格式：*

greengrass:GetDeploymentConfiguration

准許取得預先簽章的 URL 來下載大型部署文件。

當核心裝置收到指定大於 7 KB（如果部署目標為物件）或 31 KB（如果部署目標為物件群組）的部署文件時，會評估此許可。部署文件包含元件組態、部署政策和部署中繼資料。如需詳細資訊，請參閱[將 AWS IoT Greengrass 元件部署至裝置](#)。

此功能適用於 [Greengrass 核心元件](#) 的 v2.3.0 和更新版本。

資源類型：無

資源 ARN 格式：*

greengrass:ListThingGroupsForCoreDevice

准許取得核心裝置的物件群組階層。

當核心裝置從接收部署時，會檢查此許可 AWS IoT Greengrass。核心裝置使用此動作來識別自上次部署以來是否從物件群組中移除。如果核心裝置已從物件群組中移除，且該物件群組是部署到核心裝置的目標，則核心裝置會移除該部署所安裝的元件。

此功能由 [Greengrass 核心元件](#) 的 v2.5.0 和更新版本使用。

資源類型：thing (核心裝置)

資源 ARN 格式：arn:aws:iot:*region*:*account-id*:thing/*core-device-thing-name*

greengrass:VerifyClientDeviceIdentity

准許驗證連接到核心裝置的用戶端裝置的身分。

當核心裝置執行[用戶端裝置身分驗證元件](#)，並從用戶端裝置接收 MQTT 連線時，會評估此許可。用戶端裝置會呈現其 AWS IoT 裝置憑證。然後，核心裝置會將裝置憑證 AWS IoT Greengrass 傳送至雲端服務，以驗證用戶端裝置的身分。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

資源類型：無

資源 ARN 格式：*

greengrass:VerifyClientDeviceIoTCertificateAssociation

准許驗證用戶端裝置是否與 AWS IoT 憑證相關聯。

當核心裝置執行[用戶端裝置身分驗證元件](#)，並授權用戶端裝置透過 MQTT 連線時，會評估此許可。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

 Note

若要讓核心裝置使用此操作，[Greengrass 服務角色](#)必須與您的 相關聯，AWS 帳戶 並允許 `iot:DescribeCertificate` 許可。

資源類型：thing (用戶端裝置)

資源 ARN 格式：arn:aws:iot:*region*:*account-id*:thing/*client-device-thing-name*

greengrass:PutCertificateAuthorities

准許上傳用戶端裝置可下載的憑證授權單位 (CA) 憑證，以驗證核心裝置。

當核心裝置安裝並執行[用戶端裝置身分驗證元件](#)時，會評估此許可。此元件會建立本機憑證授權單位，並使用此操作來上傳其 CA 憑證。用戶端裝置使用[探索](#)操作來尋找可以連線的核心裝置時，會下載這些 CA 憑證。當用戶端裝置連線到核心裝置上的 MQTT 代理程式時，他們會使用這些 CA 憑證來驗證核心裝置的身分。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

資源類型：無

ARN 格式：*

greengrass:GetConnectivityInfo

准許取得核心裝置的連線資訊。此資訊說明用戶端裝置如何連線到核心裝置。

當核心裝置安裝並執行[用戶端裝置身分驗證元件](#)時，會評估此許可。此元件使用連線資訊來產生有效的 CA 憑證，以使用 [PutCertificateAuthorities](#) 操作上傳至 AWS IoT Greengrass 雲端服務。用戶端裝置使用這些 CA 憑證來驗證核心裝置的身分。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。

您也可以在此 AWS IoT Greengrass 控制平面上使用此操作來檢視核心裝置的連線資訊。如需詳細資訊，請參閱《AWS IoT Greengrass V1 API 參考》中的 [GetConnectivityInfo](#)。

資源類型：thing (核心裝置)

資源 ARN 格式：arn:aws:iot:*region*:*account-id*:thing/*core-device-thing-name*
greengrass:UpdateConnectivityInfo

准許更新核心裝置的連線資訊。此資訊說明用戶端裝置如何連線到核心裝置。

當核心裝置執行 [IP 偵測器元件](#) 時，會評估此許可。此元件可識別用戶端裝置連線到本機網路上核心裝置所需的資訊。然後，此元件會使用此操作將連線資訊上傳至 AWS IoT Greengrass 雲端服務，以使用戶端裝置可以使用 [探索](#) 操作擷取此資訊。如需詳細資訊，請參閱 [與本機 IoT 裝置互動](#)。

您也可以在此 AWS IoT Greengrass 控制平面上使用此操作來手動更新核心裝置的連線資訊。如需詳細資訊，請參閱 AWS IoT Greengrass V1 API 參考中的 [UpdateConnectivityInfo](#)。

資源類型：thing (核心裝置)

資源 ARN 格式：arn:aws:iot:*region*:*account-id*:thing/*core-device-thing-name*

用戶端裝置動作

greengrass:Discover

准許探索用戶端裝置可連線之核心裝置的連線資訊。此資訊說明用戶端裝置如何連線至核心裝置。用戶端裝置只能透過使用 [BatchAssociateClientDeviceWithCoreDevice](#) 操作來探索與其相關聯的核心裝置。如需詳細資訊，請參閱 [與本機 IoT 裝置互動](#)。

資源類型：thing (用戶端裝置)

資源 ARN 格式：arn:aws:iot:*region*:*account-id*:thing/*client-device-thing-name*

更新核心裝置的 AWS IoT 政策

您可以使用 AWS IoT Greengrass 和 AWS IoT 主控台或 AWS IoT API 來檢視和更新核心裝置 AWS IoT 的政策。

Note

如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，您的核心裝置具有允許存取所有 AWS IoT Greengrass 動作 () AWS IoT 的政策 `greengrass:*`。您可以依照這些步驟，限制只能存取核心裝置使用的動作。

檢閱和更新核心裝置的 AWS IoT 政策（主控台）

1. 在 [AWS IoT Greengrass 主控台](#) 導覽功能表中，選擇核心裝置。
2. 在核心裝置頁面上，選擇要更新的核心裝置。
3. 在核心裝置詳細資訊頁面上，選擇核心裝置物件的連結。此連結會在 主控台中 AWS IoT 開啟物件詳細資訊頁面。
4. 在物件詳細資訊頁面上，選擇憑證。
5. 在憑證索引標籤中，選擇物件的作用中憑證。
6. 在憑證詳細資訊頁面上，選擇政策。
7. 在政策索引標籤中，選擇要檢閱和更新 AWS IoT 的政策。您可以將必要的許可新增至連接到核心裝置作用中憑證的任何政策。

Note

如果您使用 [AWS IoT Greengrass Core 軟體安裝程式佈建資源](#)，您有兩個 AWS IoT 政策。如果 `GreengrassV2IoTThingPolicy` 存在，建議您選擇名為 `GreengrassV2IoTThingPolicy` 的政策。您使用快速安裝程式建立的核心裝置預設會使用此政策名稱。如果您將許可新增至此政策，您也會將這些許可授予使用此政策的其他核心裝置。

8. 在政策概觀中，選擇編輯作用中版本。
9. 檢閱政策，並視需要新增、移除或編輯許可。
10. 若要將新的政策版本設定為作用中版本，請在政策版本狀態下，選取將編輯的版本設定為此政策的作用中版本。
11. 選擇儲存為新版本。

檢閱和更新核心裝置 AWS IoT 的政策 (AWS CLI)

1. 列出核心裝置 AWS IoT 物件的主體。物件主體可以是 X.509 裝置憑證或其他識別符。執行下列命令，並以核心裝置的名稱取代 *MyGreengrassCore*。

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

操作會傳回回應，列出核心裝置的物件主體。

```
{
  "principals": [
    "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
  ]
}
```

2. 識別核心裝置的作用中憑證。執行下列命令，並將 *certificateId* 取代為上一個步驟中每個憑證的 ID，直到您找到作用中憑證為止。憑證 ID 是憑證 ARN 結尾的十六進位字串。--query 引數指定 僅輸出憑證的狀態。

```
aws iot describe-certificate --certificate-id certificateId --query  
'certificateDescription.status'
```

操作會以字串的形式傳回憑證狀態。例如，如果憑證處於作用中狀態，此操作會輸出 "ACTIVE"。

3. 列出連接到憑證 AWS IoT 的政策。執行下列命令，並以憑證的 ARN 取代憑證 ARN。

```
aws iot list-principal-policies --principal arn:aws:iot:us-west-2:123456789012:cert/certificateId
```

操作會傳回回應，列出連接到憑證 AWS IoT 的政策。

```
{
  "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. 選擇要檢視和更新的政策。

Note

如果您使用 [AWS IoT Greengrass Core 軟體安裝程式來佈建資源](#)，您有兩個 AWS IoT 政策。如果 GreengrassV2IoTThingPolicy 存在，我們建議您選擇名為 `GreengrassV2IoTThingPolicy` 的政策。您使用快速安裝程式建立的核心裝置預設會使用此政策名稱。如果您將許可新增至此政策，您也會將這些許可授予使用此政策的其他核心裝置。

5. 取得政策的文件。執行下列命令，並以政策的名稱取代 *GreengrassV2IoTThingPolicy*。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

操作會傳回回應，其中包含政策的文件和政策的其他資訊。政策文件是序列化為字串的 JSON 物件。

```
{
  "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. 使用線上轉換器或其他工具將政策文件字串轉換為 JSON 物件，然後將其儲存至名為 `iot-policy.json` 的檔案。

例如，如果您已安裝 [jq](#) 工具，您可以執行下列命令來取得政策文件、將其轉換為 JSON 物件，並將政策文件儲存為 JSON 物件。

```

aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
'policyDocument' | jq fromjson >> iot-policy.json

```

7. 檢閱政策文件，並視需要新增、移除或編輯許可。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來開啟檔案。

```

nano iot-policy.json

```

完成後，政策文件可能看起來類似於[核心裝置的最小 AWS IoT 政策](#)。

8. 將變更儲存為政策的新版本。執行下列命令，並以政策的名稱取代 *GreengrassV2IoTThingPolicy*。

```

aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default

```

如果成功，操作會傳回類似下列範例的回應。

```

{
  "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\\\",\\

```

```

\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"
    ],\
    \\"Resource\\": \\"*\\"
  }\
]\
}",
  "policyVersionId": "2",
  "isDefaultVersion": true
}

```

AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策

Important

較新版本的 [Greengrass 核元件](#) 需要最低 AWS IoT 政策的額外許可。您可能需要 [更新核心裝置 AWS IoT 的政策](#)，以授予其他許可。

- 當您從物件群組移除核心裝置時，執行 Greengrass 核 v2.5.0 和更新版本的核心裝置會使用 `greengrass:ListThingGroupsForCoreDevice` 許可來解除安裝元件。
- 執行 Greengrass 核 v2.3.0 和更新版本的核心裝置會使用 `greengrass:GetDeploymentConfiguration` 許可來支援大型部署組態文件。

以下範例政策包含支援核心裝置基本 Greengrass 功能所需的最少動作組合。

- 此 Connect 政策包含核心裝置物件名稱後面的 * 萬用字元（例如，`core-device-thing-name*`）。核心裝置使用相同的裝置憑證對進行多個並行訂閱 AWS IoT Core，但連線中的用戶端 ID 可能不完全符合核心裝置物件名稱。在前 50 個訂閱之後，核心裝置會使用 `core-device-thing-name#number` 做為用戶端 ID，其中每額外 50 個訂閱會 `number` 遞增。例如，當名為 `MyCoreDevice` 的核心裝置建立 150 個並行訂閱時，會使用下列用戶端 IDs：
 - 訂閱 1 到 50：MyCoreDevice
 - 訂閱 51 到 100：MyCoreDevice#2
 - 訂閱 101 到 150：MyCoreDevice#3

萬用字元允許核心裝置在使用這些尾碼為 IDs 時連線。

- 政策會列出可供核心裝置發佈訊息、訂閱和接收訊息的 MQTT 主題和主題篩選條件，包括用於陰影狀態的主題。若要支援訊息交換 AWS IoT Core、Greengrass 元件和用戶端裝置，請指定您要允許

的主題和主題篩選條件。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》[中的發佈/訂閱政策範例](#)。

- 此政策授予發佈至下列遙測資料主題的許可。

```
$aws/things/core-device-thing-name/greengrass/health/json
```

您可以針對停用遙測的核心裝置移除此許可。如需詳細資訊，請參閱[從 AWS IoT Greengrass 核心裝置收集系統運作狀態遙測資料](#)。

- 政策授予透過角色別名擔任 IAM AWS IoT 角色的許可。核心裝置使用此角色，稱為字符交換角色，以取得可用於驗證 AWS 請求的 AWS 登入資料。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

安裝 AWS IoT Greengrass Core 軟體時，您可以建立並連接第二個僅包含此許可 AWS IoT 的政策。如果您在核心裝置的主要 AWS IoT 政策中包含此許可，您可以分離和刪除其他 AWS IoT 政策。

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": "*"
    }
]
}

```

支援用戶端裝置的最低 AWS IoT 政策

下列範例政策包含支援與核心裝置上的用戶端裝置互動所需的最低動作集。若要支援用戶端裝置，核心裝置除了[基本操作的最低 AWS IoT 政策](#)之外，還必須具有此 AWS IoT 政策中的許可。

- 此政策允許核心裝置更新自己的連線資訊。只有在您將 [IP 偵測器元件](#) 部署至核心裝置時，才需要此許可 (greengrass:UpdateConnectivityInfo)。

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"
        ]
    }
]
}

```

用戶端裝置的最低 AWS IoT 政策

下列範例政策包含用戶端裝置探索核心裝置透過 MQTT 連線和通訊所需的最低動作集。用戶端裝置 AWS IoT 的政策必須包含 `greengrass:Discover` 動作，以允許裝置探索其相關聯 Greengrass 核心裝置的連線資訊。在 Resource 區段中，指定用戶端裝置的 Amazon Resource Name (ARN)，而非 Greengrass 核心裝置的 ARN。

- 此政策允許在所有 MQTT 主題上進行通訊。若要遵循最佳安全實務，請將 `iot:Publish`、`iot:Subscribe` 和 `iot:Receive` 許可限制為用戶端裝置對您的使用案例所需的一組最少主題。
- 此政策允許物件探索所有 AWS IoT 物件的核心裝置。若要遵循最佳安全實務，請將 `greengrass:Discover` 許可限制為用戶端裝置的 AWS IoT 物件或符合一組 AWS IoT 物件的萬用字元。

Important

核心裝置或 Greengrass 資料平面操作的政策中不支援的 [物件政策變數](#) (`iot:Connection.Thing.*`)。AWS IoT 反之，您可以使用與具有類似名稱的多個裝置相符的萬用字元。例如，您可以指定 `MyGreengrassDevice*` 來比對 `MyGreengrassDevice1`、`MyGreengrassDevice2` 等。

- 用戶端裝置 AWS IoT 的政策通常不需要 `iot:GetThingShadow`、`iot:UpdateThingShadow` 或 `iot:DeleteThingShadow` 動作的許可，因為 Greengrass 核心裝置會處理用戶端裝置的陰影同步操作。若要讓核心裝置能夠處理用戶端裝置影子，請檢查核心裝置 AWS IoT 的政策是否允許這些動作，以及 `Resource` 區段是否包含用戶端裝置的 ARNs。

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/*"
  ]
}
]
```

的身分和存取管理 AWS IoT Greengrass

AWS Identity and Access Management (IAM) 是一種 AWS 服務，可協助管理員安全地控制對 AWS 資源的存取。IAM 管理員可控制誰可以進行驗證（登入）和授權（具有許可）來使用 AWS IoT Greengrass 資源。IAM 是您可以免費使用 AWS 服務的。

Note

本主題說明 IAM 概念和功能。如需支援的 IAM 功能的相關資訊 AWS IoT Greengrass，請參閱 [the section called “AWS IoT Greengrass 如何使用 IAM”](#)。

目標對象

使用方式 AWS Identity and Access Management (IAM) 會根據您的角色而有所不同：

- 服務使用者 — 若無法存取某些功能，請向管理員申請所需許可 (請參閱 [對的身分和存取問題進行故障診斷 AWS IoT Greengrass](#))
- 服務管理員 — 負責設定使用者存取權並提交相關許可請求 (請參閱 [AWS IoT Greengrass 如何使用 IAM](#))
- IAM 管理員 — 撰寫政策以管理存取控制 (請參閱 [的身分型政策範例 AWS IoT Greengrass](#))

使用身分驗證

身分驗證是您 AWS 使用身分憑證登入的方式。您必須以 AWS 帳戶根使用者、IAM 使用者或擔任 IAM 角色身分進行身分驗證。

您可以使用身分來源的登入資料，例如 AWS IAM Identity Center (IAM Identity Center)、單一登入身分驗證或 Google/Facebook 登入資料，以聯合身分的形式登入。如需有關登入的詳細資訊，請參閱《AWS 登入 使用者指南》中的[如何登入您的 AWS 帳戶](#)。

對於程式設計存取，AWS 提供 SDK 和 CLI 以密碼編譯方式簽署請求。如需詳細資訊，請參閱《IAM 使用者指南》中的[API 請求的AWS 第 4 版簽署程序](#)。

AWS 帳戶 根使用者

當您建立時 AWS 帳戶，您會從一個名為 AWS 帳戶 theroot 使用者的登入身分開始，該身分具有對所有 AWS 服務和資源的完整存取權。強烈建議不要使用根使用者來執行日常任務。有關需要根使用者憑證的任務，請參閱《IAM 使用者指南》中的[需要根使用者憑證的任務](#)。

IAM 使用者和群組

IAM 使用者https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users.html是一種身分具備單人或應用程式的特定許可權。建議以臨時憑證取代具備長期憑證的 IAM 使用者。如需詳細資訊，請參閱《IAM 使用者指南》中的[要求人類使用者使用聯合身分提供者來 AWS 使用臨時憑證存取](#)。

[IAM 群組](#)會指定 IAM 使用者集合，使管理大量使用者的許可權更加輕鬆。如需詳細資訊，請參閱《IAM 使用者指南》中的[IAM 使用者的使用案例](#)。

IAM 角色

IAM 角色https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html的身分具有特定許可權，其可以提供臨時憑證。您可以透過[從使用者切換到 IAM 角色（主控台）](#)或呼叫 AWS CLI 或 AWS API 操作來擔任角色。如需詳細資訊，請參閱《IAM 使用者指南》中的[擔任角色的方法](#)。

IAM 角色適用於聯合身分使用者存取、臨時 IAM 使用者許可、跨帳戶存取權與跨服務存取，以及在 Amazon EC2 執行的應用程式。如需詳細資訊，請參閱《IAM 使用者指南》中的[IAM 中的快帳戶資源存取](#)。

使用政策管理存取權

您可以透過建立政策並將其連接到身分或資源 AWS 來控制 AWS 中的存取。政策定義與身分或資源相關聯的許可。當委託人提出請求時 AWS，會評估這些政策。大多數政策會以 JSON 文件 AWS 形式存放在中。如需進一步了解 JSON 政策文件，請參閱《IAM 使用者指南》中的[JSON 政策概觀](#)。

管理員會使用政策，透過定義哪些主體可在哪些條件下對哪些資源執行動作，以指定可存取的範圍。

預設情況下，使用者和角色沒有許可。IAM 管理員會建立 IAM 政策並將其新增至角色，供使用者後續擔任。IAM 政策定義動作的許可，無論採用何種方式執行。

身分型政策

身分型政策是附加至身分 (使用者、使用者群組或角色) 的 JSON 許可政策文件。這類政策控制身分可對哪些資源執行哪些動作，以及適用的條件。如需了解如何建立身分型政策，請參閱《IAM 使用者指南》中的[透過客戶管理政策定義自訂 IAM 許可](#)。

身分型政策可分為內嵌政策 (直接內嵌於單一身分) 與受管政策 (可附加至多個身分的獨立政策)。如需了解如何在受管政策及內嵌政策之間做選擇，請參閱《IAM 使用者指南》中的[在受管政策與內嵌政策之間選擇](#)。

資源型政策

資源型政策是附加到資源的 JSON 政策文件。範例包括 IAM 角色信任政策與 Amazon S3 儲存貯體政策。在支援資源型政策的服務中，服務管理員可以使用它們來控制對特定資源的存取權限。您必須在資源型政策中[指定主體](#)。

資源型政策是位於該服務中的內嵌政策。您無法在以資源為基礎的政策中使用來自 IAM 的 AWS 受管政策。

存取控制清單 (ACL)

存取控制清單 (ACL) 可控制哪些主體 (帳戶成員、使用者或角色) 擁有存取某資源的許可。ACL 類似於資源型政策，但它們不使用 JSON 政策文件格式。

Amazon S3 AWS WAF 和 Amazon VPC 是支援 ACLs 的服務範例。如需進一步了解 ACL，請參閱《Amazon Simple Storage Service 開發人員指南》中的[存取控制清單 \(ACL\) 概觀](#)。

其他政策類型

AWS 支援其他政策類型，可設定更多常見政策類型授予的最大許可：

- 許可界限 — 設定身分型政策可授與 IAM 實體的最大許可。如需詳細資訊，請參閱《IAM 使用者指南》中的[IAM 實體許可界限](#)。
- 服務控制政策 (SCP) — 為 AWS Organizations 中的組織或組織單位指定最大許可。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的[服務控制政策](#)。
- 資源控制政策 (RCP) — 設定您帳戶中資源可用許可的上限。如需詳細資訊，請參閱《AWS Organizations 使用者指南》中的[資源控制政策 \(RCP\)](#)。
- 工作階段政策 — 在以程式設計方式為角色或聯合身分使用者建立臨時工作階段時，以參數形式傳遞的進階政策。如需詳細資訊，請參閱《IAM 使用者指南》中的[工作階段政策](#)。

多種政策類型

當多種類型的政策適用於請求時，產生的許可會更複雜而無法理解。若要了解如何 AWS 在涉及多種政策類型時決定是否允許請求，請參閱《IAM 使用者指南》中的[政策評估邏輯](#)。

另請參閱

- [the section called “AWS IoT Greengrass 如何使用 IAM”](#)
- [the section called “身分型政策範例”](#)
- [the section called “針對識別和存取問題進行故障診斷”](#)

AWS IoT Greengrass 如何使用 IAM

在您使用 IAM 管理對的存取之前 AWS IoT Greengrass，您應該了解可與搭配使用的 IAM 功能 AWS IoT Greengrass。

IAM 功能	受到 Greengrass 支援？
具有資源層級許可的身分型政策	是
資源型政策	否
存取控制清單 (ACL)	否
標籤型授權	是
臨時憑證	是
服務連結角色	否
服務角色	是

如需其他 AWS 服務如何與 IAM 搭配使用的高階檢視，請參閱 [《AWS IAM 使用者指南》中的與 IAM 搭配使用的服務](#)。

的身分型政策 AWS IoT Greengrass

透過 IAM 身分型政策，您可以指定允許或拒絕的動作和資源，以及允許或拒絕動作的條件。AWS IoT Greengrass 支援特定動作、資源和條件索引鍵。若要了解您在政策中使用的所有元素，請參閱 [《IAM 使用者指南》中的 IAM JSON 政策元素參考](#)。

動作

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

JSON 政策的 Action 元素描述您可以用來允許或拒絕政策中存取的動作。政策會使用動作來授予執行相關聯動作的許可。

的政策動作會在動作之前 AWS IoT Greengrass 使用 greengrass: 字首。例如，若要允許某人使用 ListCoreDevices API 操作在其中列出核心裝置 AWS 帳戶，請在其政策中包含 greengrass:ListCoreDevices 動作。政策陳述式必須包含 Action 或 NotAction 元素。AWS IoT Greengrass 會定義自己的一組動作，描述您可以使用此服務執行的任務。

若要在單一陳述式中指定多個動作，請在括號 ([]) 之間列出它們，並以逗號分隔，如下所示：

```
"Action": [  
  "greengrass:action1",  
  "greengrass:action2",  
  "greengrass:action3"  
]
```

您可以使用萬用字元 (*) 來指定多個動作。例如，若要指定開頭是 List 文字的所有動作，請包含以下動作：

```
"Action": "greengrass:List*"
```

Note

建議您避免使用萬用字元來指定服務的所有可用動作。最佳實務是，您應該在政策中授予最低權限和範圍較窄的許可。如需詳細資訊，請參閱[the section called “盡可能授予最低的許可”](#)。

如需 AWS IoT Greengrass 動作的完整清單，請參閱《IAM 使用者指南》中的 [定義的動作 AWS IoT Greengrass](#)。

Resources

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Resource JSON 政策元素可指定要套用動作的物件。最佳實務是使用其 [Amazon Resource Name \(ARN\)](#) 來指定資源。若動作不支援資源層級許可，使用萬用字元 (*) 表示該陳述式適用於所有資源。

```
"Resource": "*" 
```

下表包含可用於政策陳述式的 Resource 元素中的 AWS IoT Greengrass 資源 ARN。如需 AWS IoT Greengrass 動作支援的資源層級許可映射，請參閱《IAM 使用者指南》中的 [定義的動作 AWS IoT Greengrass](#)。

某些 AWS IoT Greengrass 動作（例如，某些清單操作）無法對特定資源執行。在這些情況下，您必須使用萬用字元 (*)。

```
"Resource": "*" 
```

若要在陳述式中指定多個資源 ARNs，請在括號 ([]) 之間列出它們，並以逗號分隔，如下所示：

```
"Resource": [  
  "resource-arn1",  
  "resource-arn2",  
  "resource-arn3"  
]
```

如需 ARN 格式的詳細資訊，請參閱 [Amazon Resource Name \(ARNs\) AWS 和服務命名空間](#) Amazon Web Services 一般參考。

條件索引鍵

管理員可以使用 AWS JSON 政策來指定誰可以存取內容。也就是說，哪個主體在什麼條件下可以對什麼資源執行哪些動作。

Condition 元素會根據定義的條件，指定陳述式的執行時機。您可以建立使用 [條件運算子](#) 的條件運算式 (例如等於或小於)，來比對政策中的條件和請求中的值。若要查看所有 AWS 全域條件索引鍵，請參閱《IAM 使用者指南》中的 [AWS 全域條件內容索引鍵](#)。

範例

若要檢視 AWS IoT Greengrass 身分型政策的範例，請參閱 [the section called “身分型政策範例”](#)。

的資源型政策 AWS IoT Greengrass

AWS IoT Greengrass 不支援 [以資源為基礎的政策](#)。

存取控制清單 (ACL)

AWS IoT Greengrass 不支援 [ACLs](#)。

以 AWS IoT Greengrass 標籤為基礎的授權

您可以將標籤連接到支援 AWS IoT Greengrass 的資源，或在請求中將標籤傳遞給 AWS IoT Greengrass。若要根據標籤控制存取，您可以使用 `aws:RequestTag/${TagKey}`、`aws:ResourceTag/${TagKey}` 或 [條件索引鍵](#)，在政策的條件元素中提供標籤資訊。`aws:TagKeys` 如需詳細資訊，請參閱 [標記您的資源](#)。

的 IAM 角色 AWS IoT Greengrass

[IAM 角色](#) 是您 AWS 帳戶 中具備特定許可的實體。

搭配 使用臨時登入資料 AWS IoT Greengrass

臨時登入資料用於使用聯合身分登入、擔任 IAM 角色，或擔任跨帳戶角色。您取得暫時安全憑證的方式是透過呼叫 AWS STS API 作業 (例如，[AssumeRole](#) 或 [GetFederationToken](#))。

在 Greengrass 核心上，[裝置角色](#)的臨時登入資料可供 Greengrass 元件使用。如果您的元件使用 AWS SDK，您不需要新增邏輯來取得登入資料，因為 AWS SDK 會為您執行此操作。

服務連結角色

AWS IoT Greengrass 不支援[服務連結角色](#)。

服務角色

此功能可讓服務代表您擔任[服務角色](#)。此角色可讓服務存取其他服務中的資源，以代表您完成動作。服務角色會出現在您的 IAM 帳戶中，且由該帳戶所擁有。這表示 IAM 管理員可以變更此角色的許可。不過，這樣可能會破壞此服務的功能。

AWS IoT Greengrass 核心裝置使用服務角色來允許 Greengrass 元件和 Lambda 函數代表您存取您的部分 AWS 資源。如需詳細資訊，請參閱[the section called “授權核心裝置與 AWS 服務互動”](#)。

AWS IoT Greengrass 使用服務角色代表您存取您的部分 AWS 資源。如需詳細資訊，請參閱[Greengrass 服務角色](#)。

的身分型政策範例 AWS IoT Greengrass

根據預設，IAM 使用者和角色不具備建立或修改 AWS IoT Greengrass 資源的許可。他們也無法使用 AWS 管理主控台 AWS CLI 或 AWS API 執行任務。IAM 管理員必須建立 IAM 政策，授予使用者和角色在指定資源上執行特定 API 作業的所需許可。管理員接著必須將這些政策連接至需要這些許可的 IAM 使用者或群組。

政策最佳實務

身分型政策會判斷您帳戶中的某個人員是否可以建立、存取或刪除 AWS IoT Greengrass 資源。這些動作可能會讓您的 AWS 帳戶產生費用。當您建立或編輯身分型政策時，請遵循下列準則及建議事項：

- 開始使用 AWS 受管政策並邁向最低權限許可 – 若要開始將許可授予您的使用者和工作負載，請使用將許可授予許多常見使用案例的 AWS 受管政策。它們可在您的 中使用 AWS 帳戶。我們建議您定義特定於使用案例 AWS 的客戶受管政策，進一步減少許可。如需更多資訊，請參閱《IAM 使用者指南》中的 [AWS 受管政策](#) 或 [任務職能的 AWS 受管政策](#)。

- 套用最低權限許可 – 設定 IAM 政策的許可時，請僅授予執行任務所需的許可。為實現此目的，您可以定義在特定條件下可以對特定資源採取的動作，這也稱為最低權限許可。如需使用 IAM 套用許可的更多相關資訊，請參閱《IAM 使用者指南》中的 [IAM 中的政策和許可](#)。
- 使用 IAM 政策中的條件進一步限制存取權 – 您可以將條件新增至政策，以限制動作和資源的存取。例如，您可以撰寫政策條件，指定必須使用 SSL 傳送所有請求。如果透過特定 例如 使用服務動作 AWS 服務，您也可以使用條件來授予其存取權 CloudFormation。如需詳細資訊，請參閱《IAM 使用者指南》中的 [IAM JSON 政策元素：條件](#)。
- 使用 IAM Access Analyzer 驗證 IAM 政策，確保許可安全且可正常運作 – IAM Access Analyzer 驗證新政策和現有政策，確保這些政策遵從 IAM 政策語言 (JSON) 和 IAM 最佳實務。IAM Access Analyzer 提供 100 多項政策檢查及切實可行的建議，可協助您撰寫安全且實用的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 [使用 IAM Access Analyzer 驗證政策](#)。
- 需要多重要素驗證 (MFA) – 如果您的案例需要 IAM 使用者或 中的根使用者 AWS 帳戶，請開啟 MFA 以提高安全性。如需在呼叫 API 操作時請求 MFA，請將 MFA 條件新增至您的政策。如需詳細資訊，請參閱《IAM 使用者指南》中的 [透過 MFA 的安全 API 存取](#)。

如需 IAM 中最佳實務的相關資訊，請參閱《IAM 使用者指南》中的 [IAM 安全最佳實務](#)。

政策範例

下列客戶定義的政策範例會授予常見案例的許可。

範例

- [允許使用者檢視他們自己的許可](#)

若要了解如何使用這些範例 JSON 政策文件建立 IAM 身分型政策，請參閱《IAM 使用者指南》中的 [在 JSON 標籤上建立政策](#)。

允許使用者檢視他們自己的許可

此範例會示範如何建立政策，允許 IAM 使用者檢視附加到他們使用者身分的內嵌及受管政策。此政策包含在主控台或使用 或 AWS CLI AWS API 以程式設計方式完成此動作的許可。

```
{
  "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": "*"
    }
]
}

```

授權核心裝置與 AWS 服務互動

AWS IoT Greengrass 核心裝置使用 AWS IoT Core 登入資料提供者來授權對 AWS 服務的呼叫。AWS IoT Core 登入資料提供者可讓裝置使用其 X.509 憑證做為唯一裝置身分來驗證 AWS 請求。這不需要在 AWS IoT Greengrass 核心裝置上存放 AWS 存取金鑰 ID 和私密存取金鑰。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權直接呼叫 AWS 服務](#)。

當您執行 AWS IoT Greengrass Core 軟體時，您可以選擇佈建核心裝置所需的 AWS 資源。這包括核心裝置透過 AWS IoT Core 登入資料提供者擔任的 AWS Identity and Access Management (IAM) 角色。使用 `--provision true` 引數來設定角色和政策，以允許核心裝置取得臨時 AWS 登入資料。此引數也會設定指向此 IAM AWS IoT 角色的角色別名。您可以指定要使用的 IAM 角色和 AWS IoT 角色別名的名稱。如果您在 `--provision true` 沒有這些其他名稱參數的情況下指定，Greengrass 核心裝置會建立並使用下列預設資源：

- IAM 角色：GreengrassV2TokenExchangeRole

此角色具有名為 `GreengrassV2TokenExchangeRoleAccess` 以及允許 `credentials.iot.amazonaws.com` 擔任該角色的信任關係。此政策包含核心裝置的最低許可。

 Important

此政策不包含對 S3 儲存貯體中檔案的存取。您必須將許可新增至角色，以允許核心裝置從 S3 儲存貯體擷取元件成品。如需詳細資訊，請參閱 [允許存取元件成品的 S3 儲存貯體](#)。

- AWS IoT 角色別名：`GreengrassV2TokenExchangeRoleAlias`

此角色別名是指 IAM 角色。

如需詳細資訊，請參閱 [步驟 3：安裝 AWS IoT Greengrass 核心軟體](#)。

您也可以設定現有核心裝置的角色別名。若要這樣做，請設定 [Greengrass 核元件](#) 的 `iotRoleAlias` 組態參數。

您可以取得此 IAM 角色的臨時 AWS 登入資料，以在自訂元件中執行 AWS 操作。如需詳細資訊，請參閱 [與 AWS 服務互動](#)。

主題

- [核心裝置的服務角色許可](#)
- [允許存取元件成品的 S3 儲存貯體](#)

核心裝置的服務角色許可

該角色允許下列服務擔任該角色：

- `credentials.iot.amazonaws.com`

如果您使用 AWS IoT Greengrass 核心軟體建立此角色，它會使用下列許可政策來允許核心裝置連接和傳送日誌 AWS。政策的名稱預設為以結尾的 IAM 角色名稱 `Access`。例如，如果您使用預設 IAM 角色名稱，則此政策的名稱為 `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": "*"
    }
  ]
}
```

允許存取元件成品的 S3 儲存貯體

預設的核心裝置角色不允許核心裝置存取 S3 儲存貯體。若要部署在 S3 儲存貯體中具有成品的元件，您必須新增允許核心裝置下載元件成品的 `s3:GetObject` 許可。您可以將新政策新增至核心裝置角色，以授予此許可。

新增允許存取 Amazon S3 中元件成品的政策

1. 建立名為 `component-artifact-policy.json` 的檔案，並將下列 JSON 複製到 檔案。此政策允許存取 S3 儲存貯體中的所有檔案。將 `amzn-s3-demo-bucket` 取代為 S3 儲存貯體的名稱，以允許核心裝置存取。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
    }
  ]
}
```

2. 執行下列命令，從 中的政策文件建立政策 `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
```

從輸出中的政策中繼資料複製政策 Amazon Resource Name (ARN)。您可以在下一個步驟中使用此 ARN 將此政策連接至核心裝置角色。

3. 執行下列命令，將政策連接至核心裝置角色。以您在執行 AWS IoT Greengrass Core 軟體時指定的角色名稱取代 *GreengrassV2TokenExchangeRole*。然後，將政策 ARN 取代為上一個步驟的 ARN。

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
```

如果命令沒有輸出，則會成功，而且您的核心裝置可以存取您上傳至此 S3 儲存貯體的成品。

安裝程式佈建資源的最低 IAM 政策

安裝 AWS IoT Greengrass Core 軟體時，您可以佈建必要的 AWS 資源，例如裝置的 AWS IoT 物件和 IAM 角色。您也可以將本機開發工具部署至裝置。安裝程式需要 AWS 登入資料，才能在您的 中執行這些動作 AWS 帳戶。如需詳細資訊，請參閱[安裝 AWS IoT Greengrass Core 軟體](#)。

下列範例政策包含安裝程式佈建這些資源所需的最低動作集。如果您指定安裝程式的 --provision 引數，則需要這些許可。將 *account-id* 取代為您的 AWS 帳戶 ID，並將 *GreengrassV2TokenExchangeRole* 取代為您使用--tes-role-name[安裝程式引數](#)指定的字符交換角色名稱。

 Note

只有在您指定安裝程式的 `--deploy-dev-tools` 引數時，才需要 `DeployDevTools` 政策陳述式。

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": "*"    
  }  
]  
}
```

Greengrass 服務角色

Greengrass 服務角色是 AWS Identity and Access Management (IAM) 服務角色，授權代表您 AWS IoT Greengrass 存取來自 AWS 服務的資源。此角色可讓 AWS IoT Greengrass 驗證用戶端裝置的身分，並管理核心裝置連線資訊。

Note

AWS IoT Greengrass V1 也會使用此角色來執行基本任務。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的 [Greengrass 服務角色](#)。

若要允許 AWS IoT Greengrass 存取您的資源，Greengrass 服務角色必須與相關聯，AWS 帳戶並指定 AWS IoT Greengrass 為信任的實體。此角色必須包含 [AWSGreengrassResourceAccessRolePolicy](#) 受管政策或自訂政策，該政策會為您使用 AWS IoT Greengrass 的功能定義同等許可。AWS 會維護此政策，此政策會定義 AWS IoT Greengrass 用於存取 AWS 資源的一組許可。如需詳細資訊，請參閱 [AWS 受管政策：AWSGreengrassResourceAccessRolePolicy](#)。

您可以重複使用相同的 Greengrass 服務角色 AWS 區域，但必須在您使用的每個 AWS 區域中將其與您的帳戶建立關聯 AWS IoT Greengrass。如果服務角色未在目前的 [中](#) 設定 AWS 區域，核心裝置將無法驗證用戶端裝置，也無法更新連線資訊。

下列各節說明如何使用 AWS 管理主控台 或 建立和管理 Greengrass 服務角色 AWS CLI。

主題

- [管理 Greengrass 服務角色（主控台）](#)
- [管理 Greengrass 服務角色 \(CLI\)](#)
- [另請參閱](#)

 Note

除了授權服務層級存取的服務角色之外，您還可以將權杖交換角色指派給 Greengrass 核心裝置。字串交換角色是獨立的 IAM 角色，可控制核心裝置上的 Greengrass 元件和 Lambda 函數如何存取 AWS 服務。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

管理 Greengrass 服務角色（主控台）

AWS IoT 主控台可讓您輕鬆管理您的 Greengrass 服務角色。例如，當您設定核心裝置的用戶端裝置探索時，主控台會檢查是否 AWS 帳戶連接到目前中的 Greengrass 服務角色 AWS 區域。如果未連接，主控台可以為您建立和設定服務角色。如需詳細資訊，請參閱[the section called “建立 Greengrass 服務角色”](#)。

您可以使用 主控台執行下列角色管理任務：

主題

- [尋找您的 Greengrass 服務角色 \(主控台\)](#)
- [建立 Greengrass 服務角色 \(主控台\)](#)
- [變更 Greengrass 服務角色 \(主控台\)](#)
- [分離 Greengrass 服務角色 \(主控台\)](#)

 Note

登入主控台的使用者必須擁有可檢視、建立或變更服務角色的許可。

尋找您的 Greengrass 服務角色 (主控台)

使用下列步驟來尋找目前 AWS IoT Greengrass 中使用的服務角色 AWS 區域。

1. 導覽至 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇設定。
3. 捲動至 Greengrass service role (Greengrass 服務角色) 區段，查看您的服務角色及其政策。

如果您沒有看到服務角色，主控台可以為您建立或設定。如需詳細資訊，請參閱[建立 Greengrass 服務角色](#)。

建立 Greengrass 服務角色 (主控台)

主控台可以為您建立和設定預設的 Greengrass 服務角色。這個角色具有以下屬性：

屬性	Value
名稱	Greengrass_ServiceRole
信任實體	AWS service: greengrass
政策	AWSGreengrassResourceAccessRolePolicy (AWSGreengrassResourceAccessRolePolicy)

Note

如果您使用 [AWS IoT Greengrass V1 裝置設定指令碼](#) 建立此角色，則角色名稱為 `GreengrassServiceRole_`*random-string*。

當您設定核心裝置的用戶端裝置探索時，主控台會檢查 Greengrass 服務角色是否與目前 AWS 帳戶中的相關聯 AWS 區域。如果沒有，主控台會提示您允許代表您 AWS IoT Greengrass 讀取和寫入 AWS 服務。

如果您授予許可，主控台會檢查 Greengrass_ServiceRole 中是否存在名為的角色 AWS 帳戶。

- 如果角色存在，主控台會將服務角色連接到目前 AWS 帳戶中的 AWS 區域。
- 如果角色不存在，主控台會建立預設 Greengrass 服務角色，並將其連接到目前 AWS 帳戶中的 AWS 區域。

Note

如果您想要使用自訂角色政策建立服務角色，請使用 IAM 主控台來建立或修改角色。如需詳細資訊，請參閱 [《IAM 使用者指南》中的建立角色以將許可委派給 AWS 服務或修改角色](#)。請確定角色會授與與您所使用功能和資源之 `AWSGreengrassResourceAccessRolePolicy` 受管政策相同的許可。我們建議您也在信任政策中包含 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵

會限制存取，只允許來自指定帳戶和 Greengrass 工作區的這些請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

如果您建立服務角色，請返回 AWS IoT 主控台，並將該角色連接到您的 AWS 帳戶。您可以在設定頁面上的 Greengrass 服務角色下執行此操作。

變更 Greengrass 服務角色 (主控台)

使用下列程序選擇不同的 Greengrass 服務角色，以在主控台 AWS 區域 目前選取的 AWS 帳戶 中連接至您的 。

1. 導覽至 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇設定。
3. 在 Greengrass 服務角色下，選擇變更角色。

更新 Greengrass 服務角色對話方塊隨即開啟，並顯示 中定義為 AWS IoT Greengrass 信任實體 AWS 帳戶 的 IAM 角色。

4. 選擇要連接的 Greengrass 服務角色。
5. 選擇連接角色。

分離 Greengrass 服務角色 (主控台)

使用下列程序，在目前的 中將 Greengrass 服務角色從 AWS 您的帳戶分離 AWS 區域。這會撤銷 AWS IoT Greengrass 存取目前 AWS 服務的權限 AWS 區域。

Important

分離服務角色可能會中斷作用中的操作。

1. 導覽至 [AWS IoT 主控台](#)。
2. 在導覽窗格中，選擇設定。
3. 在 Greengrass 服務角色下，選擇分離角色。
4. 在確認對話方塊中，選擇 Detach (分離)。

Note

如果您不再需要該角色，您可以在 IAM 主控台中刪除該角色。如需詳細資訊，請參閱 IAM 使用者指南中的[刪除角色或執行個體描述檔](#)。

其他角色可能允許 AWS IoT Greengrass 存取您的資源。若要尋找允許代表您 AWS IoT Greengrass 擔任許可的所有角色，請在 IAM 主控台的角色頁面上，尋找包含 AWS 服務的角色：信任實體欄中的 greengrass。

管理 Greengrass 服務角色 (CLI)

在下列程序中，我們假設 AWS Command Line Interface 已安裝並設定為使用您的 AWS 帳戶。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的[安裝、更新和解除安裝 AWS CLI](#)和[設定 AWS CLI](#)。

您可以使用 AWS CLI 執行下列角色管理任務：

主題

- [取得 Greengrass 服務角色 \(CLI\)](#)
- [建立 Greengrass 服務角色 \(CLI\)](#)
- [移除 Greengrass 服務角色 \(CLI\)](#)

取得 Greengrass 服務角色 (CLI)

使用下列程序來了解 Greengrass 服務角色是否與 AWS 帳戶中的相關聯 AWS 區域。

- 取得服務角色。將##取代為您的 AWS 區域（例如，us-west-2）。

```
aws greengrassv2 get-service-role-for-account --region region
```

如果 Greengrass 服務角色已與您的帳戶建立關聯，則請求會傳回下列角色中繼資料。

```
{
  "associatedAt": "timestamp",
  "roleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

如果請求未傳回角色中繼資料，則必須建立服務角色（如果不存在），並將其與 中的帳戶建立關聯 AWS 區域。

建立 Greengrass 服務角色 (CLI)

使用下列步驟來建立角色，並將其與您的 建立關聯 AWS 帳戶。

使用 IAM 建立服務角色

1. 使用允許 AWS IoT Greengrass 擔任角色的信任政策來建立角色。此範例會建立名為 Greengrass_ServiceRole 的角色，但您可以使用不同的名稱。我們建議您也在信任政策中包含 aws:SourceArn 和 aws:SourceAccount 全域條件內容金鑰，以協助防止混淆代理人安全問題。條件內容索引鍵會限制存取，只允許來自指定帳戶和 Greengrass 工作區的這些請求。如需有關混淆代理人問題的詳細資訊，請參閱 [預防跨服務混淆代理人](#)。

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. 從輸出中的角色中繼資料，複製角色 ARN。您使用 ARN 將角色與您的帳戶相關聯。
3. 將 AWSGreengrassResourceAccessRolePolicy 政策連接到角色。

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

將服務角色與 建立關聯 AWS 帳戶

- 將角色與您的帳戶相關聯。將 *role-arn* 取代為服務角色 ARN，並將 *region* 取代為 your AWS 區域（例如，us-west-2）。

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn --  
region region
```

如果成功，請求會傳回下列回應。

```
{  
  "associatedAt": "timestamp"  
}
```

移除 Greengrass 服務角色 (CLI)

使用下列步驟取消 Greengrass 服務角色與 的關聯 AWS 帳戶。

- 將服務角色與您的帳戶取消關聯。將 *##* 取代為您的 AWS 區域（例如，us-west-2）。

```
aws greengrassv2 disassociate-service-role-from-account --region region
```

如果成功，會傳回下列回應。

```
{  
  "disassociatedAt": "timestamp"  
}
```

Note

如果您未在任何 中使用服務角色，您應該將其刪除 AWS 區域。首先，使用 [delete-role-policy](#) 將 AWSGreengrassResourceAccessRolePolicy 受管政策從角色中分離，然後使用 [delete-role](#) 來刪除角色。如需詳細資訊，請參閱 IAM 使用者指南中的 [刪除角色或執行個體描述檔](#)。

另請參閱

- 《IAM 使用者指南》中的[建立角色以將許可委派給 AWS 服務](#)
- IAM 使用者指南中的[修改角色](#)
- 《IAM 使用者指南》中的[刪除角色或執行個體描述檔](#)
- AWS IoT Greengrass 命令AWS CLI 參考中的 命令
 - [associate-service-role-to-account](#)
 - [disassociate-service-role-from-account](#)
 - [get-service-role-for-account](#)
- 《命令AWS CLI 參考》中的 IAM 命令
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

AWS 的 受管政策 AWS IoT Greengrass

AWS 受管政策是由 AWS AWS 受管政策建立和管理的獨立政策旨在為許多常用案例提供許可，以便您可以開始將許可指派給使用者、群組和角色。

請記住，AWS 受管政策可能不會授予特定使用案例的最低權限許可，因為這些許可可供所有 AWS 客戶使用。我們建議您定義特定於使用案例的[客戶管理政策](#)，以便進一步減少許可。

您無法變更 AWS 受管政策中定義的許可。如果 AWS 更新受 AWS 管政策中定義的許可，則更新會影響政策連接的所有委託人身分（使用者、群組和角色）。當新的 AWS 服務 啟動或新的 API 操作可用於現有服務時，AWS 最有可能更新 AWS 受管政策。

如需詳細資訊，請參閱 IAM 使用者指南中的 [AWS 受管政策](#)。

主題

- [AWS 受管政策：AWSGreengrassFullAccess](#)
- [AWS 受管政策：AWSGreengrassReadOnlyAccess](#)
- [AWS 受管政策：AWSGreengrassResourceAccessRolePolicy](#)
- [AWS IoT Greengrass AWS 受管政策的更新](#)

AWS 受管政策：AWSGreengrassFullAccess

您可將 AWSGreengrassFullAccess 政策連接到 IAM 身分。

此政策會授予管理許可，允許委託人完整存取所有 AWS IoT Greengrass 動作。

許可詳細資訊

此政策包含以下許可：

- greengrass – 允許主體完整存取所有 AWS IoT Greengrass 動作。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS 受管政策：AWSGreengrassReadOnlyAccess

您可將 AWSGreengrassReadOnlyAccess 政策連接到 IAM 身分。

此政策授予唯讀許可，允許委託人檢視但不修改其中的資訊 AWS IoT Greengrass。例如，具有這些許可的主體可以檢視部署到 Greengrass 核心裝置的元件清單，但無法建立部署來變更在該裝置上執行的元件。

許可詳細資訊

此政策包含以下許可：

- greengrass – 允許主體執行傳回項目清單或項目詳細資訊的動作。這包括以 List 或 開頭的 API 操作 Get。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:List*",
        "greengrass:Get*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS 受管政策：AWSGreengrassResourceAccessRolePolicy

您可以將AWSGreengrassResourceAccessRolePolicy政策連接至 IAM 實體。AWS IoT Greengrass 也會將此政策連接至允許代表您 AWS IoT Greengrass 執行動作的服務角色。如需詳細資訊，請參閱[Greengrass 服務角色](#)。

此政策授予管理許可，AWS IoT Greengrass 允許執行基本任務，例如擷取 Lambda 函數、管理 AWS IoT 裝置影子，以及驗證 Greengrass 用戶端裝置。

許可詳細資訊

此政策包含以下許可。

- greengrass – 管理 Greengrass 資源。
- iot (*Shadow) – 管理名稱中具有下列特殊識別符的 AWS IoT 陰影。需要這些許可，AWS IoT Greengrass 才能與核心裝置通訊。
 - *-gci – AWS IoT Greengrass 使用此影子來存放核心裝置連線資訊，讓用戶端裝置可以探索並連線至核心裝置。
 - *-gcm – AWS IoT Greengrass V1 使用此影子來通知核心裝置 Greengrass 群組的憑證授權單位 (CA) 憑證已輪換。
 - *-gda – AWS IoT Greengrass V1 使用此影子來通知核心裝置部署。
 - GG_* – 未使用。

- `iot` (`DescribeThing` 和 `DescribeCertificate`) – 擷取 AWS IoT 物件和憑證的相關資訊。需要這些許可，AWS IoT Greengrass 才能驗證連線至核心裝置的用戶端裝置。如需詳細資訊，請參閱[與本機 IoT 裝置互動](#)。
- `lambda` – 擷取 AWS Lambda 函數的相關資訊。需要此許可，V AWS IoT Greengrass V1 才能將 Lambda 函數部署到 Greengrass 核心。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的[在 AWS IoT Greengrass 核心上執行 Lambda 函數](#)。
- `secretsmanager` – 擷取名稱開頭為 `greengrass-` 之 AWS Secrets Manager 秘密的值。需要此許可，V AWS IoT Greengrass V1 才能將 Secrets Manager 秘密部署到 Greengrass 核心。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的[將秘密部署至 AWS IoT Greengrass 核心](#)。
- `s3` – 從名稱包含 `greengrass` 或 `sagemaker` 的 S3 儲存貯體擷取檔案物件。需要這些許可，V AWS IoT Greengrass V1 才能部署存放在 S3 儲存貯體中的機器學習資源。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的[機器學習資源](#)。
- `sagemaker` – 擷取 Amazon SageMaker AI 機器學習推論模型的相關資訊。需要此許可，V AWS IoT Greengrass V1 才能將 ML 模型部署到 Greengrass 核心。如需詳細資訊，請參閱《AWS IoT Greengrass V1 開發人員指南》中的[執行機器學習推論](#)。

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 AWS 受管政策的更新

從此服務開始追蹤這些變更起 AWS IoT Greengrass，您可以檢視 AWS 受管政策更新的詳細資訊。如需此頁面變更的自動提醒，請訂閱 [AWS IoT Greengrass V2 文件歷史記錄頁面上](#) 的 RSS 摘要。

變更	描述	Date
AWS IoT Greengrass 開始追蹤變更	AWS IoT Greengrass 已開始追蹤其 AWS 受管政策的變更。	2021 年 7 月 2 日

預防跨服務混淆代理人

混淆代理人問題屬於安全性議題，其中沒有執行動作許可的實體可以強制具有更多許可的實體執行該動作。在中 AWS，跨服務模擬可能會導致混淆代理人問題。在某個服務 (呼叫服務) 呼叫另一個服務 (被呼叫服務) 時，可能會發生跨服務模擬。可以操縱呼叫服務來使用其許可，以其不應有存取許可的方式對其他客戶的資源採取動作。為了預防這種情況，AWS 提供的工具可協助您保護所有服務的資料，而這些服務主體已獲得您帳戶中資源的存取權。

我們建議在資源政策中使用 [aws:SourceArn](#) 和 [aws:SourceAccount](#) 全域條件內容索引鍵，以限制將另一個服務 AWS IoT Greengrass 提供給資源的許可。如果同時使用全域條件內容金鑰，則在相同政策陳述式中使用 `aws:SourceAccount` 值和 `aws:SourceArn` 值中的帳戶時，必須使用相同的帳戶 ID。

的值 `aws:SourceArn` 必須是與 `sts:AssumeRole` 請求相關聯的 Greengrass 客戶資源。

防範混淆代理人問題最有效的方法，是使用 `aws:SourceArn` 全域條件內容金鑰，以及資源的完整 ARN。如果不知道資源的完整 ARN，或者如果您指定了多個資源，請使用 `aws:SourceArn` 全域條件內容金鑰，同時使用萬用字元 (*) 表示 ARN 的未知部分。例如 `arn:aws:greengrass::account-id:*`。

如需使用 `aws:SourceArn` 和 `aws:SourceAccount` 全域條件內容索引鍵的政策範例，請參閱 [建立 Greengrass 服務角色](#)。

對的身分和存取問題進行故障診斷 AWS IoT Greengrass

使用以下資訊來協助您診斷和修正使用 AWS IoT Greengrass 和 IAM 時可能遇到的常見問題。

問題

- [我未獲授權在中執行動作 AWS IoT Greengrass](#)
- [我未獲授權執行 iam:PassRole](#)
- [我是管理員，想要允許其他人存取 AWS IoT Greengrass](#)

- [我想要允許以外的人員 AWS 帳戶 存取我的 AWS IoT Greengrass 資源](#)

如需一般的故障診斷協助，請參閱 [疑難排解](#)。

我未獲授權在 中執行動作 AWS IoT Greengrass

若您收到指出您未獲授權執行動作的錯誤，您必須聯絡管理員以取得協助。您的管理員是提供您使用者名稱和密碼的人員。

當 IAM mateojackson 使用者嘗試檢視核心裝置的詳細資訊，但沒有 greengrass:GetCoreDevice 許可時，會發生下列範例錯誤。

```
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
```

在此情況下，Mateo 會請求管理員更新他的政策，允許他使用 arn:aws:greengrass:us-west-2:123456789012:coreDevices/MyGreengrassCore 動作存取 greengrass:GetCoreDevice 資源。

以下是使用時可能遇到的一般 IAM 問題 AWS IoT Greengrass。

我未獲授權執行 iam:PassRole

如果您收到錯誤，告知您未獲授權執行 iam:PassRole 動作，您的政策必須更新，允許您將角色傳遞給 AWS IoT Greengrass。

有些 AWS 服務可讓您將現有角色傳遞給該服務，而不是建立新的服務角色或服務連結角色。如需執行此作業，您必須擁有將角色傳遞至該服務的許可。

名為 marymajor 的 IAM 使用者嘗試使用主控台在 AWS IoT Greengrass 中執行動作時，發生下列範例錯誤。但是，動作要求服務具備服務角色授予的許可。Mary 沒有將角色傳遞給服務的許可。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

在這種情況下，Mary 的政策必須更新，允許她執行 iam:PassRole 動作。

如果您需要協助，請聯絡您的 AWS 管理員。您的管理員提供您的簽署憑證。

我是管理員，想要允許其他人存取 AWS IoT Greengrass

若要允許其他人存取 AWS IoT Greengrass，您必須將許可授予需要存取的人員或應用程式。如果您使用 AWS IAM Identity Center 管理人員和應用程式，您可以將許可集指派給使用者或群組，以定義其存取層級。許可集會自動建立 IAM 政策，並將其指派給與該人員或應用程式相關聯的 IAM 角色。如需詳細資訊，請參閱 AWS IAM Identity Center 《使用者指南》中的[許可集](#)。

如果您不是使用 IAM Identity Center，則必須為需要存取的人員或應用程式建立 IAM 實體（使用者或角色）。您接著必須將政策連接到實體，在 AWS IoT Greengrass 中授予他們正確的許可。授予許可後，請將登入資料提供給使用者或應用程式開發人員。他們將使用這些登入資料來存取 AWS。若要進一步了解如何建立 IAM 使用者、群組、政策和許可，請參閱《IAM [使用者指南](#)》中的 [IAM 身分和政策](#)和許可。

我想要允許以外的人員 AWS 帳戶 存取我的 AWS IoT Greengrass 資源

您可以建立 IAM 角色，讓其他帳戶中的使用者或您組織外部的人員可用來存取您的 AWS 資源。您可以指定要允許哪些信任對象擔任該角色。如需詳細資訊，請參閱《IAM [使用者指南](#)》中的[提供存取權給您 AWS 帳戶 擁有的另一個 IAM 使用者](#)，以及[提供存取權給第三方 AWS 帳戶擁有的](#)。

AWS IoT Greengrass 不支援以資源型政策或存取控制清單 (ACLs) 為基礎的跨帳戶存取。

允許透過代理或防火牆的裝置流量

Greengrass 核心裝置和 Greengrass 元件會對 AWS 服務和其他網站執行傳出請求。作為安全措施，您可以將傳出流量限制在少量端點和連接埠。您可以使用下列有關端點和連接埠的資訊，透過代理、防火牆或 [Amazon VPC 安全群組](#) 限制裝置流量。如需如何設定核心裝置以使用代理的詳細資訊，請參閱[連線至連接埠 443 或透過網路代理](#)。

主題

- [基本操作的端點](#)
- [使用自動佈建進行安裝的端點](#)
- [AWS 所提供元件的端點](#)

基本操作的端點

Greengrass 核心裝置使用下列端點和連接埠進行基本操作。

擷取 AWS IoT 端點

取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置使用這些端點來連線 AWS IoT。請執行下列操作：

1. 取得 AWS IoT 的資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 取得的 AWS IoT 登入資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-  
west-2.amazonaws.com"
}
```

Endpoint	站點	必要	Description
greengrass-ats.iot . <i>region</i> .amazonaws.com	8443 或 443	是	用於資料平面操作，例如安裝部署和使用用戶端裝置。
<i>device-data-prefix</i> -ats.iot. <i>region</i> .amazonaws.com	MQTT : 8883 或 443 HTTPS : 8443 或 443	是	用於裝置管理的資料平面操作，例如 MQTT

Endpoint	站點	必要	Description
			通訊和陰影 同步 AWS IoT Core。
<i>device-credentials</i> <i>-prefix</i> .credentials.iot. <i>region</i> .amazonaws.com	443	是	用來取得 AWS 登入 資料，核心 裝置會使用 此登入資料 從 Amazon S3 下載元 件成品並 執行其他操 作。如需詳 細資訊，請 參閱 授權核 心裝置與 AWS 服務 互動 。
*.s3.amazonaws.com *.s3. <i>region</i> .amazonaws.com	443	是	用於部署。 此格式包含 * 字元，因 為端點字首 是在內部控 制，可能隨 時變更。

Endpoint	站點	必要	Description
<code>data.iot.<i>region</i>.amazonaws.com</code>	443	否	如果核心裝置執行比 v2.4.0 更舊的 Greengrass 核心版本 ，且設定為使用網路代理，則為必要項目。核心裝置在代理後 AWS IoT Core 使用此端點與進行 MQTT 通訊。如需詳細資訊，請參閱 設定網路代理 。

使用自動佈建進行安裝的端點

當您[使用自動資源佈建安裝 AWS IoT Greengrass Core 軟體時](#)，Greengrass 核心裝置會使用下列端點和連接埠。

Endpoint	站點	必要	Description
<code>iot.<i>region</i>.amazonaws.com</code>	443	是	用來建立 AWS IoT 資源並擷取現有 AWS IoT 資源的相關資訊。

Endpoint	站點	必要	Description
iam.amazonaws.com	443	是	用來建立 IAM 資源並擷取現有 IAM 資源的相關資訊。
sts. <i>region</i> .amazonaws.com	443	是	用來取得的 ID AWS 帳戶。
greengrass. <i>region</i> .amazonaws.com	443	否	如果您使用 --deploy-dev-tools 引數將 Greengrass CLI 元件部署到核心裝置，則為必要。

AWS所提供元件的端點

Greengrass 核心裝置使用其他端點，取決於其執行的軟體元件。您可以在本開發人員指南中每個元件頁面的要求區段中找到每個 AWS 所提供元件所需的端點。如需詳細資訊，請參閱[AWS提供的元件](#)。

的合規驗證 AWS IoT Greengrass

若要了解 AWS 服務 是否在特定合規計劃範圍內，請參閱[AWS 服務 合規計劃範圍內](#)然後選擇您感興趣的合規計劃。如需一般資訊，請參閱[AWS 合規計劃](#)。

您可以使用 下載第三方稽核報告 AWS Artifact。如需詳細資訊，請參閱[下載報告下載 AWS Artifact](#)。

您使用 時的合規責任 AWS 服務 取決於資料的機密性、您公司的合規目標，以及適用的法律和法規。如需使用 時合規責任的詳細資訊 AWS 服務，請參閱 [AWS 安全文件](#)。

FIPS 端點

AWS IoT Greengrass 支援使用 FIPS ([聯邦資訊處理標準 \(FIPS\) 140-2](#)) 端點。啟用 FIPS 模式時，所有資料傳輸，包括 HTTP 和 MQTT 通訊協定，到 AWS 雲端服務都應叫用並與對應的 FIPS 相容端點 ([FIPS - Amazon Web Services \(AWS\)](#)) 建立連線。

MQTT 通訊 AWS IoT 會使用 IoT 資料平面 FIPS 端點 ([連線至 AWS IoT FIPS 端點 - AWS IoT Core](#)) 和 AWS 開發的 FIPS 相容密碼編譯程式庫 `aws-lc`。

針對 Greengrass 中的 HTTP 通訊：

- 對於核和外掛程式元件，所有 SDK HTTP 用戶端都會透過將系統屬性 `AWS_USE_FIPS_ENDPOINT` 設定為 `true` 來設定 FIPS 端點；
- 對於一般元件，所有元件從系統屬性 `AWS_USE_FIPS_ENDPOINT` 設定為 `true` 開始。此程序可確保這些一般元件所使用的 SDK HTTP 用戶端會將請求傳送至 FIPS 相容端點。

Note

如果是串流管理員，Nucleus 會傳遞環境變數 `AWS_GG_FIPS_MODE`。此環境變數可讓串流管理員內使用的 HTTP 用戶端識別並連線至對應的 FIPS 相容端點。

AWS IoT Greengrass 提供兩種啟用 FIPS 模式的方法：佈建和部署。若要啟用 FIPS 模式，您必須將組態參數 `fipsMode` 設定為 `true`、Nucleus 接著將系統屬性 `AWS_USE_FIPS_ENDPOINT` 設定為 `true`，並將其做為環境變數傳播至所有其他元件。此外，AWS IoT Greengrass 將下載根 CA 憑證 (CA3)，並將其附加至現有的 `rootCA.pem` (或 `AmazonRootCA1.pem`)。如果您透過新部署啟用 FIPS，Nucleus 會重新啟動，以確保系統屬性在啟用 FIPS 模式後生效。

除了設定 `fipsMode` 參數之外，您還必須設定 `iotDataEndpoint`、`iotCredEndpoint` 和 `greengrassDataEndpoint` 參數。如需詳細資訊，請參閱下列相關文件。

使用部署啟用 FIPS 端點

取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置使用這些端點來連線 AWS IoT。需要兩個端點：`iotDataEndpoint` 和 `iotCredEndpoint`。請執行下列操作：

- 在 FIPS 資料 [AWS IoT Core 平面端點中取得您區域的 FIPS 資料端點](#)。您的 FIPS 資料端點看起來 AWS 帳戶 應該如下所示：`https://data.iot-fips.us-west-2.amazonaws.com`

2. 在 FIPS [AWS IoT Core 資料平面端點](#)中取得您區域的 FIPS 登入資料端點。您的 FIPS 登入資料端點看起來 AWS 帳戶 應該如下所示：`https://data.credentials.iot-fips.us-west-2.amazonaws.com`

然後，若要啟用具有部署的 FIPS，您需要將下列組態套用至「Nucleus」。部署上要合併的組態如下。

Console

要合併的組態

```
{
  "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

下列命令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

dashboard-deployment.json 檔案包含下列 JSON 文件。

```
{
  "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

下列 [Greengrass CLI](#) 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \  
  --recipeDir recipes \  
  --artifactDir artifacts \  
  --merge "aws.greengrass.Nucleus=2.13.0" \  
  --update-config dashboard-configuration.json
```

dashboard-configuration.json 檔案包含下列 JSON 文件。

```
{  
  "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"  
    }  
  }  
}
```

使用手動資源佈建搭配 FIPS 端點安裝 Nucleus

使用 FIPS 端點手動佈建 AWS IoT Greengrass V2 核心裝置 AWS 的資源

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [擷取 AWS IoT 端點](#)
- [建立 AWS IoT 物件](#)
- [建立物件憑證](#)
- [建立權杖交換角色](#)

- [將憑證下載至裝置](#)
- [設定裝置環境](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)

擷取 AWS IoT 端點

取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置使用這些端點來連線 AWS IoT。需要兩個端點：`iotDataEndpoint`和 `iotCredEndpoint`。請執行下列操作：

1. 在 FIPS 資料[AWS IoT Core 平面端點](#)中取得您區域的 FIPS 資料端點。您的 FIPS 資料端點看起來 AWS 帳戶 應該如下所示：`https://data.iot-fips.us-west-2.amazonaws.com`
2. 在 FIPS [AWS IoT Core 資料平面端點](#)中取得您區域的 FIPS 登入資料端點。您的 FIPS 登入資料端點看起來 AWS 帳戶 應該如下所示：`https://data.credentials.iot-fips.us-west-2.amazonaws.com`

建立 AWS IoT 物件

AWS IoT 物件代表連線至 的裝置和邏輯實體 AWS IoT。Greengrass 核心裝置是 AWS IoT 實物。當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。

在本節中，您會建立代表裝置的 AWS IoT 物件。

建立 AWS IoT 物件

1. 為您的裝置建立 AWS IoT 物件。在您的開發電腦上，執行下列命令。
 - 將 *MyGreengrassCore* 取代為要使用的物件名稱。此名稱也是 Greengrass 核心裝置的名稱。

Note

物件名稱不能包含冒號 (:) 字元。

```
aws iot create-thing --thing-name MyGreengrassCore
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (選用) 將 AWS IoT 物件新增至新的或現有的物件群組。您可以使用物件群組來管理 Greengrass 核心裝置的機群。當您將軟體元件部署到裝置時，您可以將個別裝置或裝置群組設為目標。您可以將裝置新增至具有作用中 Greengrass 部署的物件群組，以將該物件群組的軟體元件部署到裝置。請執行下列操作：

- a. (選用) 建立 AWS IoT 物件群組。

- 將 *MyGreengrassCoreGroup* 取代為要建立的物件群組名稱。

 Note

物件群組名稱不能包含冒號 (:) 字元。

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

- b. 將 AWS IoT 物件新增至物件群組。

- 將 *MyGreengrassCore* 取代為您 AWS IoT 物件的名稱。
- 將 *MyGreengrassCoreGroup* 取代為物件群組的名稱。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-name MyGreengrassCoreGroup
```

如果請求成功，命令不會有任何輸出。

建立物件憑證

當您將裝置註冊為 AWS IoT 物件時，該裝置可以使用數位憑證進行身分驗證 AWS。此憑證可讓裝置與 AWS IoT 和 通訊 AWS IoT Greengrass。

在本節中，您可以建立和下載裝置可用來連線的憑證 AWS。

如果您想要將 AWS IoT Greengrass Core 軟體設定為使用硬體安全模組 (HSM) 來安全地存放私有金鑰和憑證，請依照步驟從 HSM 中的私有金鑰建立憑證。否則，請依照步驟在 AWS IoT 服務中建立憑證和私有金鑰。硬體安全功能僅適用於 Linux 裝置。如需硬體安全性和使用需求的詳細資訊，請參閱 [硬體安全整合](#)。

在 AWS IoT 服務中建立憑證和私有金鑰

建立物件憑證

1. 建立資料夾以下載 AWS IoT 物件的憑證。

```
mkdir greengrass-v2-certs
```

2. 建立並下載 AWS IoT 物件的憑證。

```
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
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/  
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificateId":  
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",  
  "certificatePem": "-----BEGIN CERTIFICATE-----  
MIICiTCCAfICCCQD6m7oRw0uX0jANBgkqhkiG9w  
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAGTA1dBMRAdDgYDVQQHEwdTZ  
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2x1MRIw  
EAYDVQQDEw1UZXN0Q21sYWMxHzAdBgkqhkiG9w0BCQEWEG5vb251QGftYXpvi5
```

```

jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBA5TC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXR0Q21sYWVx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGfYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySwTc2XADZ4nB+BLyGVI
k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcVQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEATCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwax1Aoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQ8AMICgKCAQEAEEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEYjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqX0gHh/xJTwo
+sGpWEXAMPLEDz18x0d2ka4tCzuWEXAMPLEeahJbYkCPUBSU8opVkr7qkEXAMPLE1DR6sx2Hocli00Ltu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEEv9mQ0UXP6p1fgxwKRX2fEXAMPLEda\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}

```

儲存憑證的 Amazon Resource Name (ARN)，以供稍後用於設定憑證。

從 HSM 中的私有金鑰建立憑證

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

建立物件憑證

1. 在核心裝置上，初始化 HSM 中的 PKCS#11 字符，並產生私有金鑰。私有金鑰必須是 RSA-2048 金鑰大小（或更大）或 ECC 金鑰的 RSA 金鑰。

Note

若要搭配 ECC 金鑰使用硬體安全模組，您必須使用 [Greengrass nucleus](#) v2.5.6 或更新版本。

若要使用硬體安全模組和[秘密管理員](#)，您必須搭配 RSA 金鑰使用硬體安全模組。

檢查您的 HSM 文件，了解如何初始化字符並產生私有金鑰。如果您的 HSM 支援物件 IDs，請在產生私有金鑰時指定物件 ID。儲存您在初始化字符和產生私有金鑰時指定的槽 ID、使用者 PIN、物件標籤、物件 ID（如果您的 HSM 使用）。當您將物件憑證匯入 HSM 並設定 AWS IoT Greengrass Core 軟體時，稍後會使用這些值。

2. 從私有金鑰建立憑證簽署請求 (CSR)。AWS IoT 使用此 CSR 為您在 HSM 中產生的私有金鑰建立物件憑證。如需有關如何從私有金鑰建立 CSR 的資訊，請參閱 HSM 的文件。CSR 是檔案，例如 `iotdevicekey.csr`。
3. 將 CSR 從裝置複製到您的開發電腦。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸 CSR。將 *device-ip-address* 取代為您裝置的 IP 地址，並將 *~/iotdevicekey.csr* 取代為裝置上的 CSR 檔案路徑。

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 在開發電腦上，建立您下載 AWS IoT 物件憑證的資料夾。

```
mkdir greengrass-v2-certs
```

5. 使用 CSR 檔案來建立物件 AWS IoT 的憑證，並將其下載到您的開發電腦。

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-request=file:///iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/device.pem.crt
```

如果請求成功，回應看起來與下列範例類似。

```
{
```

```

"certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
"certificateId":
"aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
"certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uX0jANBgkqhkiG9w
0BAQUFADCBIDELMAKGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXN0Q2l5YWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTEwNDI1MjA0NTIxWjCBiDELMAKGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2x1MRIwEAYDVQQDEw1UZXN0Q2l5YWMx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wgZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLygVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcvcQAaRHhd1QWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJI1J00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp3780D8uTs7fLvJx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}

```

儲存憑證的 ARN，以供稍後用於設定憑證。

接著，設定物件憑證。如需詳細資訊，請參閱[設定物件憑證](#)。

建立權杖交換角色

Greengrass 核心裝置使用稱為字符交換角色的 IAM 服務角色來授權呼叫 AWS 服務。裝置使用 AWS IoT 登入資料提供者取得此角色的臨時 AWS 登入資料，允許裝置與 互動 AWS IoT、將日誌傳送至 Amazon CloudWatch Logs，以及從 Amazon S3 下載自訂元件成品。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

您可以使用 AWS IoT 角色別名來設定 Greengrass 核心裝置的字符交換角色。角色別名可讓您變更裝置的字符交換角色，但讓裝置組態保持不變。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[授權直接呼叫 AWS 服務](#)。

在本節中，您會建立權杖交換 IAM 角色和指向角色 AWS IoT 的角色別名。如果您已經設定 Greengrass 核心裝置，您可以使用其權杖交換角色和角色別名，而不是建立新的權杖交換角色和角色別名。然後，您將裝置的 AWS IoT 物件設定為使用該角色和別名。

建立權杖交換 IAM 角色

1. 建立您的裝置可用作字符交換角色的 IAM 角色。請執行下列操作：

- a. 建立檔案，其中包含權杖交換角色所需的信任政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-trust-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 使用信任政策文件建立權杖交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為要建立的 IAM 角色名稱。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AR0AZ2YMUHYHK50KM77FB",
    "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. 建立檔案，其中包含權杖交換角色所需的存取政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano device-role-access-policy.json
```

將下列 JSON 複製到 檔案。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

 Note

此存取政策不允許存取 S3 儲存貯體中的元件成品。若要在 Amazon S3 中部署定義成品的自訂元件，您必須將許可新增至角色，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

如果您還沒有元件成品的 S3 儲存貯體，您可以在建立儲存貯體之後新增這些許可。

d. 從政策文件中建立 IAM 政策。

- 以要建立的 IAM 政策名稱取代 *GreengrassV2TokenExchangeRoleAccess*。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --  
policy-document file://device-role-access-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{  
  "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. 將 IAM 政策連接至字符交換角色。

- 將 *GreengrassV2TokenExchangeRole* 取代為 IAM 角色的名稱。
- 將政策 ARN 取代為您在上一個步驟中建立的 IAM 政策 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果請求成功，命令不會有任何輸出。

2. 建立指向字符交換 AWS IoT 角色的角色別名。

- 以要建立的角色別名名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
- 將角色 ARN 取代為您在上一個步驟中建立之 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

Note

若要建立角色別名，您必須具有將字符交換 IAM 角色傳遞至 的許可 AWS IoT。如果您在嘗試建立角色別名時收到錯誤訊息，請檢查您的 AWS 使用者是否具有此許可。如需詳細資訊，請參閱AWS Identity and Access Management 《使用者指南》中的[授予使用者將角色傳遞至 AWS 服務的許可](#)。

3. 建立並連接允許 Greengrass 核心裝置使用角色別名來擔任字符交換角色 AWS IoT 的政策。如果您之前已設定 Greengrass 核心裝置，您可以連接其角色別名 AWS IoT 政策，而不是建立新的。請執行下列操作：

- a. （選用）建立檔案，其中包含角色別名所需的 AWS IoT 政策文件。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano greengrass-v2-iot-role-alias-policy.json
```

將下列 JSON 複製到 檔案。

- 使用角色別名的 ARN 取代資源 ARN。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. 從 AWS IoT 政策文件中建立政策。

- 將 *GreengrassCoreTokenExchangeRoleAliasPolicy* 取代為要建立 AWS IoT 的政策名稱。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "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. 將 AWS IoT 政策連接至 AWS IoT 物件的憑證。

- 以角色別名 AWS IoT 政策的名稱取代 *GreengrassCoreTokenExchangeRoleAliasPolicy*。
- 將目標 ARN 取代為物件 AWS IoT 憑證的 ARN。

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
--target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果請求成功，命令不會有任何輸出。

將憑證下載至裝置

先前，您已將裝置的憑證下載到開發電腦。在本節中，您將憑證複製到核心裝置，以使用裝置用來連線的憑證來設定裝置 AWS IoT。您也可以下載 Amazon 根憑證授權機構 (CA) 憑證。如果您使用 HSM，您也可以將憑證檔案匯入本節的 HSM。

- 如果您先前已在 AWS IoT 服務中建立物件憑證和私有金鑰，請依照步驟下載具有私有金鑰和憑證檔案的憑證。
- 如果您先前從硬體安全模組 (HSM) 中的私有金鑰建立物件憑證，請依照步驟下載 HSM 中具有私有金鑰和憑證的憑證。

下載具有私有金鑰和憑證檔案的憑證

將憑證下載至裝置

1. 將 AWS IoT 物件憑證從開發電腦複製到裝置。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸憑證。將 *device-ip-address* 取代為您裝置的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 在裝置上建立 Greengrass 根資料夾。您稍後會將 AWS IoT Greengrass 核心軟體安裝到此資料夾。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 C:\greengrass\v2 或 D:\greengrass\v2，將 Greengrass 元件路徑保持在 260 個字元的限制以下。

Linux or Unix

- */greengrass/v2* 將取代為要使用的資料夾。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 *C#\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 *C#\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

3. (僅限 Linux) 設定 Greengrass 根資料夾父項的許可。

- 將 */greengrass* 取代為根資料夾的父系。

```
sudo chmod 755 /greengrass
```

4. 將 AWS IoT 物件憑證複製到 Greengrass 根資料夾。

Linux or Unix

- `/greengrass/v2` 將取代為 Greengrass 根資料夾。

```
sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
```

Windows Command Prompt

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
```

PowerShell

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
```

5. 下載 Amazon 根憑證授權機構 (CA) 憑證。AWS IoT 憑證預設會與 Amazon 根憑證授權機構憑證相關聯。下載 CA1 憑證和 [CA3 憑證](#)。

Linux or Unix

- 將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 Greengrass 根資料夾。

```
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
```

在 HSM 中使用私有金鑰和憑證下載憑證

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

將憑證下載至裝置

1. 將 AWS IoT 物件憑證從開發電腦複製到裝置。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸憑證。將 *device-ip-address* 取代為您裝置的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

2. 在裝置上建立 Greengrass 根資料夾。您稍後會將 AWS IoT Greengrass 核心軟體安裝到此資料夾。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 `C:\greengrass\v2` 或 `D:\greengrass\v2`，將 Greengrass 元件路徑保持在 260 個字元的限制以下。

Linux or Unix

- */greengrass/v2* 將取代為要使用的資料夾。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 `C#\greengrass\v2` 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

- (僅限 Linux) 設定 Greengrass 根資料夾父項的許可。

- 將 `/greengrass` 取代為根資料夾的父系。

```
sudo chmod 755 /greengrass
```

- 將物件憑證檔案 匯入 `~/greengrass-v2-certs/device.pem.crt`HSM。檢查 HSM 的文件，了解如何將憑證匯入其中。使用您先前在 HSM 中產生私有金鑰的相同字符、槽 ID、使用者 PIN、物件標籤和物件 ID (如果您的 HSM 使用) 匯入憑證。

Note

如果您先前在沒有物件 ID 的情況下產生私有金鑰，且憑證具有物件 ID，請將私有金鑰的物件 ID 設定為與憑證相同的值。檢查您的 HSM 文件，了解如何設定私有金鑰物件的物件 ID。

- (選用) 刪除物件憑證檔案，使其僅存在於 HSM 中。

```
rm ~/greengrass-v2-certs/device.pem.crt
```

- 下載 Amazon 根憑證授權機構 (CA) 憑證。AWS IoT 憑證預設會與 Amazon 根憑證授權機構憑證相關聯。同時下載 CA1 和 [CA3 憑證](#)。

Linux or Unix

- 將 `/greengrass/v2` 或 `C#\greengrass\v2` 取代為 Greengrass 根資料夾。

```
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
```

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

- 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2 :

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023 :

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成後，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. (選用) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者 (通常為 root) 具有使用任何使用者和任何群組執行 `sudo` 的許可。

- a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. (選用) 若要[執行容器化 Lambda 函數](#)，您必須啟用 [cgroups](#) v1，而且必須啟用和掛載記憶體和裝置 cgroups。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱您作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 中的需求清單所示[裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.0 和更新版本。

為 設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始選單搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。

- 在 Windows 裝置的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是 [wmic 已棄用命令](#) 的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name = 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

- 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
- 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將##取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 Accept 同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將##取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```

下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為的檔案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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#)之規定。

2. （選用）驗證 Greengrass 核軟體簽章

 Note

此功能適用於 Greengrass 核版本 2.9.5 及更新版本。

- a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 調用會產生輸出，指出驗證的結果。
- i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：

```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。

3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

如果您安裝的 Greengrass 核版本早於 2.4.0 版，安裝 AWS IoT Greengrass 核心軟體後請勿移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass Core 軟體後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從部分組態檔案安裝，指定 AWS 使用您先前建立的資源和憑證。Core AWS IoT Greengrass 軟體使用組態檔案，指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案建立完整的組態檔案。

- 指定 以使用 `ggc_user` 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 `ggc_group` 系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 [使用 JVM 選項控制記憶體配置](#)。

- 如果您先前已在 AWS IoT 服務中建立物件憑證和私有金鑰，請依照步驟安裝具有私有金鑰和憑證檔案的 AWS IoT Greengrass 核心軟體。
- 如果您先前從硬體安全模組 (HSM) 中的私有金鑰建立物件憑證，請依照步驟在 HSM 中安裝具有私有金鑰和憑證 AWS IoT Greengrass 的 Core 軟體。

使用私有金鑰和憑證檔案安裝 AWS IoT Greengrass 核心軟體

安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass Core 軟體的版本。

- 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到 檔案。此部分組態檔案會指定系統參數和 Greengrass 核參數。

```
---
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"
```

然後，執行下列動作：

- 將 的每個執行個體取代 */greengrass/v2* 為 Greengrass 根資料夾。
 - 將 *MyGreengrassCore* 取代為 AWS IoT 物件的名稱。
 - 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
 - 將 *us-west-2* 取代 AWS 區域 為您建立資源的 。
 - 以字符交換角色別名的名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
 - 將 *iotDataEndpoint* 取代為您的 AWS IoT 資料端點。
 - 將 *iotCredEndpoint* 取代為您的 AWS IoT 登入資料端點。
3. 執行安裝程式，並指定 `--init-config` 以提供組態檔案。
- 將 */greengrass/v2* 或 *C#\greengrass\v2* 取代為 Greengrass 根資料夾。
 - 將 *GreengrassInstaller* 的每個執行個體取代為您解壓縮安裝程式的資料夾。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需有關直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

4. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 `config`、`packages` 和 `logs`。

在 HSM 中使用私有金鑰和憑證安裝 AWS IoT Greengrass 核心軟體

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.5.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 核心裝置上的此功能。

安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass Core 軟體的版本。
 - 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

- 若要讓 AWS IoT Greengrass Core 軟體在 HSM 中使用私有金鑰和憑證，請在安裝 AWS IoT Greengrass Core 軟體時安裝 [PKCS#11 提供者元件](#)。PKCS#11 供應商元件是您可以在安裝期間設定的外掛程式。您可以從下列位置下載最新版本的 PKCS#11 提供者元件：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>

將 PKCS#11 供應商外掛程式下載至名為 `aws.greengrass.crypto.Pkcs11Provider.jar` 的檔案。

將 `aws.greengrass.crypto.Pkcs11Provider.jar` 將 *GreengrassInstaller* 取代為您要使用的資料夾。

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

- 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到檔案。此部分組態檔案會指定系統參數、Greengrass 核參數和 PKCS#11 提供者參數。

```
---
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"
```

然後，執行下列動作：

- 將 PKCS#11 URIs 中的每個 *iotdevicekey* 執行個體取代為您建立私有金鑰並匯入憑證的物件標籤。
 - 將的每個執行個體取代 */greengrass/v2* 為 Greengrass 根資料夾。
 - 將 *MyGreengrassCore* 取代為 AWS IoT 物件的名稱。
 - 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
 - 將 *us-west-2* 取代 AWS 區域 為您建立資源的。
 - 以字符交換角色別名的名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
 - 將 取代 *iotDataEndpoint* 為 AWS IoT 您的資料端點。
 - 將 取代 *iotCredEndpoint* 為您的 AWS IoT 登入資料端點。
 - 將 *aws.greengrass.crypto.Pkcs11Provider* 元件的組態參數取代為核心裝置上的 HSM 組態值。
4. 執行安裝程式，並指定 *--init-config* 以提供組態檔案。
- */greengrass/v2* 將 取代為 Greengrass 根資料夾。
 - 將 *GreengrassInstaller* 的每個執行個體取代為您解壓縮安裝程式的資料夾。

```
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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

ℹ Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需有關直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

5. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 `config`、`packages` 和 `logs`。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱 [執行 AWS IoT Greengrass 核心軟體](#)。

如需如何設定和使用軟體的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

使用機群佈建安裝 FIPS 端點

此功能適用於 [Greengrass 核心元件](#) 的 v2.4.0 和更新版本。

在 AWS IoT Greengrass Core 軟體上安裝 FIPS 端點，並為您的核心裝置佈建 AWS IoT 機群。

Note

機群佈建外掛程式目前不支援在硬體安全模組 (HSM) 中存放私有金鑰和憑證檔案。若要使用 HSM，[請使用手動佈建安裝 AWS IoT Greengrass 核心軟體](#)。

若要使用機 AWS IoT 群佈建安裝 AWS IoT Greengrass 核心軟體，您必須在 [中設定 AWS IoT AWS 帳戶](#) 用來佈建 Greengrass 核心裝置的資源。這些資源包括佈建範本、宣告憑證和[字符交換 IAM 角色](#)。建立這些資源之後，您可以重複使用這些資源，以在機群中佈建多個核心裝置。如需詳細資訊，請參閱[設定 Greengrass 核心裝置的 AWS IoT 機群佈建](#)。

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [先決條件](#)
- [擷取 AWS IoT 端點](#)
- [將憑證下載至裝置](#)
- [設定裝置環境](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [下載 AWS IoT 機群佈建外掛程式](#)

- [安裝 AWS IoT Greengrass 核心軟體](#)

先決條件

若要使用機 AWS IoT 群佈建安裝 AWS IoT Greengrass 核心軟體，您必須先[為 Greengrass 核心裝置設定 AWS IoT 機群佈建](#)。完成這些步驟一次後，您可以使用機群佈建，在任何數量的裝置上安裝 AWS IoT Greengrass Core 軟體。

擷取 AWS IoT 端點

取得您的 FIPS 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置使用這些端點來連線 AWS IoT。請執行下列操作：

1. 在 FIPS 資料[AWS IoT Core 平面端點](#)中取得您區域的 FIPS 資料端點。您的 FIPS 資料端點看起來 AWS 帳戶 應該如下所示：`https://data.iot-fips.us-west-2.amazonaws.com`
2. 在 FIPS [AWS IoT Core 資料平面端點](#)中取得您區域的 FIPS 登入資料端點。您的 FIPS 登入資料端點看起來 AWS 帳戶 應該如下所示：`https://data.credentials.iot-fips.us-west-2.amazonaws.com`

將憑證下載至裝置

裝置使用宣告憑證和私有金鑰來驗證其佈建 AWS 資源和取得 X.509 裝置憑證的請求。您可以在製造期間將宣告憑證和私有金鑰嵌入裝置，或在安裝期間將憑證和金鑰複製到裝置。在本節中，您將宣告憑證和私有金鑰複製到裝置。您也可以將 Amazon 根憑證授權機構 (CA) 憑證下載到裝置。

Important

佈建宣告私有金鑰應隨時受到保護，包括在 Greengrass 核心裝置上。我們建議您使用 Amazon CloudWatch 指標和日誌來監控是否有濫用跡象，例如未經授權使用宣告憑證來佈建裝置。如果您偵測到濫用，請停用佈建宣告憑證，使其無法用於裝置佈建。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[監控 AWS IoT](#)。

為了協助您更有效管理在中自行註冊的裝置數量和裝置 AWS 帳戶，您可以在建立機群佈建範本時指定預先佈建掛鉤。預先佈建掛鉤是一種 AWS Lambda 函數，可驗證裝置在註冊期間提供的範本參數。例如，您可以建立預先佈建掛鉤，檢查資料庫的裝置 ID，以確認裝置具有佈建的許可。如需詳細資訊，請參閱《AWS IoT Core 開發人員指南》中的[預先佈建掛鉤](#)。

將宣告憑證下載至裝置

1. 將宣告憑證和私有金鑰複製到裝置。如果在開發電腦和裝置上啟用 SSH 和 SCP，您可以使用開發電腦上的 `scp` 命令來傳輸宣告憑證和私有金鑰。下列範例命令 `claim-certs` 會將開發電腦上名為的資料夾傳輸至裝置。將 *device-ip-address* 取代為您裝置的 IP 地址。

```
scp -r claim-certs/ device-ip-address:~
```

2. 在裝置上建立 Greengrass 根資料夾。您稍後會將 AWS IoT Greengrass 核心軟體安裝到此資料夾。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 `C:\greengrass\v2` 或 `D:\greengrass\v2`，將 Greengrass 元件路徑保持在 260 個字元的限制以下。

Linux or Unix

- */greengrass/v2* 將 取代為要使用的資料夾。

```
sudo mkdir -p /greengrass/v2
```

Windows Command Prompt

- 將 *C#\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

PowerShell

- 將 *C#\greengrass\v2* 取代為要使用的資料夾。

```
mkdir C:\greengrass\v2
```

3. (僅限 Linux) 設定 Greengrass 根資料夾父項的許可。

- 將 `/greengrass` 取代為根資料夾的父系。

```
sudo chmod 755 /greengrass
```

4. 將宣告憑證移至 Greengrass 根資料夾。

- 將 `/greengrass/v2` 或 `C:\greengrass\v2` 取代為 Greengrass 根資料夾。

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. 同時下載 CA1 憑證和 [CA3 憑證](#)。

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
```

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成後，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. (選用) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者 (通常為 `root`) 具有使用任何使用者和任何群組執行 `sudo` 的許可。
 - a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. (選用) 若要[執行容器化 Lambda 函數](#)，您必須啟用 [cgroups](#) v1，而且必須啟用和掛載記憶體和裝置 `cgroups`。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 `cgroups` 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱您作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 [中的需求清單](#)所示[裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核元件](#) 的 v2.5.0 和更新版本。

為設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
 4. 在 Windows 裝置的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是 [wmic 已棄用命令](#) 的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

- 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
- 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將 **##** 取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 以 Accept 同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將 **##** 取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```


下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為 `greengrass-nucleus-latest.zip` 的檔案。

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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

2. (選用) 驗證 Greengrass 核軟體簽章

Note

此功能適用於 Greengrass 核版本 2.9.5 及更新版本。

- a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。`jdk17.0.6_10` 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 調用會產生輸出，指出驗證的結果。
 - i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：

```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。
3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

Important

如果您安裝的 Greengrass 核心版本早於 2.4.0 版，安裝 AWS IoT Greengrass 核心軟體後請勿移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass Core 軟體後移除此資料夾。

下載 AWS IoT 機群佈建外掛程式

您可以從下列位置下載最新版本的 AWS IoT 機群佈建外掛程式：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar>

Note

您可以從下列位置下載特定版本的 AWS IoT 機群佈建外掛程式。將##取代為要下載的版本。如需每個機群佈建外掛程式版本的詳細資訊，請參閱 [AWS IoT 機群佈建外掛程式 changelog](#)。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-  
FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
```

機群佈建外掛程式是開放原始碼。若要檢視其原始程式碼，請參閱 GitHub 上的 [AWS IoT 機群佈建外掛程式](#)。

下載 AWS IoT 機群佈建外掛程式

- 在您的裝置上，將 AWS IoT 機群佈建外掛程式下載至名為 `aws.greengrass.FleetProvisioningByClaim.jar` 的檔案夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

安裝 AWS IoT Greengrass 核心軟體

使用指定下列動作的引數執行安裝程式：

- 從指定使用機群佈建外掛程式佈建 AWS 資源的部分組態檔案安裝。Core AWS IoT Greengrass 軟體使用組態檔案，指定裝置上每個 Greengrass 元件的組態。安裝程式會從您提供的部分組態檔案和機群佈建外掛程式建立 AWS 的資源建立完整的組態檔案。
- 指定以使用 `ggc_user` 系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用 `ggc_group` 系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

⚠ Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

📘 Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱 [使用 JVM 選項控制記憶體配置](#)。

安裝 AWS IoT Greengrass 核心軟體

1. 檢查 AWS IoT Greengrass Core 軟體的版本。

- 將 *GreengrassInstaller* 取代為包含軟體之資料夾的路徑。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到檔案。此部分組態檔案會指定機群佈建外掛程式的參數。如需您可以指定之選項的詳細資訊，請參閱 [設定 AWS IoT 機群佈建外掛程式](#)。

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"

```

然後，執行下列動作：

- 將 *2.16.1* 取代為 AWS IoT Greengrass 核心軟體版本。
- 將 */greengrass/v2* 或 *C#\greengrass\v2* 的每個執行個體取代為 Greengrass 根資料夾。

Note

在 Windows 裝置上，您必須將路徑分隔符號指定為雙反斜線 (\\)，例如 C:\\greengrass\\v2。

- 將 *us-west-2* 取代為您建立佈建範本和其他資源 AWS 的區域。
 - 將 *iotDataEndpoint* 取代為您的 AWS IoT 資料端點。
 - 將 *iotCredentialEndpoint* 取代為您的 AWS IoT 登入資料端點。
 - 以字符交換角色別名的名稱取代 *GreengrassCoreTokenExchangeRoleAlias*。
 - 以機群佈建範本的名稱取代 *GreengrassFleetProvisioningTemplate*。
 - 將 *claimCertificatePath* 取代為裝置上的宣告憑證路徑。
 - 將 *claimCertificatePrivateKeyPath* 取代為裝置上的宣告憑證私有金鑰路徑。
 - 將範本參數 (templateParameters) 取代為用來佈建裝置的值。此範例是指定義 ThingName 和 ThingGroupName 參數 [的範例範本](#)。
3. 執行安裝程式。指定 *--trusted-plugin* 以提供機群佈建外掛程式，並指定 *--init-config* 提供組態檔案。
- */greengrass/v2* 將 取代為 Greengrass 根資料夾。
 - 將 *GreengrassInstaller* 的每個執行個體取代為您解壓縮安裝程式的資料夾。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為系統服務 `Successfully set up Nucleus as a system service` 時列印。否則，如果安裝程式成功安裝軟體，就不會輸出任何訊息。

Note

當您在沒有 `deploy-dev-tools` 引數的情況下執行安裝程式時，無法使用 `--provision true` 引數來部署本機開發工具。如需有關直接在您的裝置上部署 Greengrass CLI 的資訊，請參閱 [Greengrass 命令列界面](#)。

4. 透過檢視根資料夾中的檔案來驗證安裝。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安裝成功，根資料夾會包含數個資料夾，例如 config、packages 和 logs。

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)。

如需如何設定和使用軟體的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

使用自動資源佈建安裝 FIPS 端點

Core AWS IoT Greengrass 軟體包含安裝程式，可將您的裝置設定為 Greengrass 核心裝置。若要快速設定裝置，安裝程式可以佈建核心裝置操作所需的 AWS IoT 物件、AWS IoT 物件群組、IAM 角色和 AWS IoT 角色別名。安裝程式也可以將本機開發工具部署到核心裝置，因此您可以使用裝置來開發和測試自訂軟體元件。安裝程式需要 AWS 登入資料才能佈建這些資源並建立部署。

如果您無法提供 AWS 登入資料給裝置，您可以佈建 AWS 核心裝置操作所需的資源。您也可以將開發工具部署到核心裝置，以用作開發裝置。這可讓您在執行安裝程式時為裝置提供較少的許可。如需詳細資訊，請參閱[使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

Important

下載 AWS IoT Greengrass Core 軟體之前，請檢查您的核心裝置是否符合安裝和執行 AWS IoT Greengrass Core 軟體 v2.0 [的需求](#)。

主題

- [設定裝置環境](#)
- [提供 AWS 登入資料給裝置](#)
- [下載 AWS IoT Greengrass 核心軟體](#)
- [安裝 AWS IoT Greengrass 核心軟體](#)

設定裝置環境

請依照本節中的步驟設定 Linux 或 Windows 裝置，以用作您的 AWS IoT Greengrass 核心裝置。

設定 Linux 裝置

設定的 Linux 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。下列命令說明如何在裝置上安裝 OpenJDK。

- 針對 Debian 為基礎或 Ubuntu 為基礎的發行版本：

```
sudo apt install default-jdk
```

- 針對 Red Hat 為基礎的發行版本：

```
sudo yum install java-11-openjdk-devel
```

- 針對 Amazon Linux 2：

```
sudo amazon-linux-extras install java-openjdk11
```

- 針對 Amazon Linux 2023：

```
sudo dnf install java-11-amazon-corretto -y
```

安裝完成時，請執行下列命令來驗證 Java 是否在您的 Linux 裝置上執行。

```
java -version
```

命令會列印在裝置上執行的 Java 版本。例如，在 Debian 型分佈上，輸出看起來可能會類似下列範例。

```
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. (選用) 建立在裝置上執行元件的預設系統使用者和群組。您也可以選擇讓 AWS IoT Greengrass Core 軟體安裝程式在安裝期間使用 `--component-default-user` 安裝程式引數建立此使用者和群組。如需詳細資訊，請參閱[安裝程式引數](#)。

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. 確認執行 AWS IoT Greengrass 核心軟體的使用者 (通常為 `root`) 具有使用任何使用者和任何群組執行 `sudo` 的許可。

- a. 執行下列命令以開啟 `/etc/sudoers` 檔案。

```
sudo visudo
```

- b. 確認使用者的許可如下範例所示。

```
root    ALL=(ALL:ALL) ALL
```

4. (選用) 若要[執行容器化 Lambda 函數](#)，您必須啟用 [cgroups](#) v1，而且必須啟用和掛載記憶體和裝置 cgroups。如果您不打算執行容器化 Lambda 函數，您可以略過此步驟。

若要啟用這些 cgroups 選項，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

如需有關檢視和設定裝置核心參數的資訊，請參閱作業系統和開機載入器的文件。依照指示永久設定核心參數。

5. 在裝置上安裝所有其他必要的相依性，如 中的需求清單所示 [裝置要求](#)。

設定 Windows 裝置

Note

此功能適用於 [Greengrass 核心元件](#) 的 v2.5.0 和更新版本。

為 設定 Windows 裝置 AWS IoT Greengrass V2

1. 安裝執行 AWS IoT Greengrass 核心軟體所需的 Java 執行時間。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。
2. 檢查 [PATH](#) 系統變數上是否提供 Java，如果沒有，請新增它。LocalSystem 帳戶會執行 AWS IoT Greengrass Core 軟體，因此您必須將 Java 新增至 PATH 系統變數，而不是使用者的 PATH 使用者變數。請執行下列操作：
 - a. 按 Windows 鍵開啟開始選單。
 - b. 輸入 **environment variables** 以從開始功能表搜尋系統選項。
 - c. 在開始功能表搜尋結果中，選擇編輯系統環境變數以開啟系統屬性視窗。
 - d. 選擇環境變數...以開啟環境變數視窗。
 - e. 在系統變數下，選取路徑，然後選擇編輯。在編輯環境變數視窗中，您可以檢視個別行上的每個路徑。
 - f. 檢查 Java 安裝bin資料夾的路徑是否存在。路徑看起來可能會類似下列範例。

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

- g. 如果路徑中缺少 Java 安裝的bin資料夾，請選擇新增以新增，然後選擇確定。
3. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
 4. 在 Windows 裝置上的 LocalSystem 帳戶中建立預設使用者。將##取代為安全密碼。

```
net user /add ggc_user password
```

 Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是已[wmic棄用命令](#)的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =  
'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

- 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
- 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將##取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

如果 PsExec License Agreement 開啟，請選擇 以Accept同意授權並執行 命令。

 Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將預設使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

提供 AWS 登入資料給裝置

將 AWS 登入資料提供給您的裝置，讓安裝程式可以佈建所需的 AWS 資源。如需所需許可的詳細資訊，請參閱[安裝程式佈建資源的最低 IAM 政策](#)。

提供 AWS 登入資料給裝置

- 將 AWS 登入資料提供給裝置，讓安裝程式可以為您的核心裝置佈建 AWS IoT 和 IAM 資源。為了提高安全性，我們建議您取得 IAM 角色的臨時登入資料，只允許佈建所需的最低許可。如需詳細資訊，請參閱[安裝程式佈建資源的最低 IAM 政策](#)。

Note

安裝程式不會儲存或存放您的登入資料。

在您的裝置上，執行下列其中一項操作來擷取登入資料，並將其提供給 AWS IoT Greengrass Core 軟體安裝程式：

- (建議) 使用來自的臨時登入資料 AWS IAM Identity Center
 - 從 IAM Identity Center 提供存取金鑰 ID、私密存取金鑰和工作階段字符。如需詳細資訊，請參閱《IAM Identity Center 使用者指南》中的[取得和重新整理暫時憑證](#)中的手動憑證重新整理。
 - 執行下列命令，將登入資料提供給 AWS IoT Greengrass 核心軟體。

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- 使用來自 IAM 角色的臨時安全登入資料：
 - a. 提供您擔任之 IAM 角色的存取金鑰 ID、私密存取金鑰和工作階段字符。如需如何擷取這些登入資料的詳細資訊，請參閱《IAM 使用者指南》中的[請求臨時安全登入](#)資料。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass 核心軟體。

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY  
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE  
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY  
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"  
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY"  
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o50ytwEXAMPLE="
```

- 使用來自 IAM 使用者的長期登入資料：
 - a. 為您的 IAM 使用者提供存取金鑰 ID 和私密存取金鑰。您可以建立 IAM 使用者來佈建您稍後刪除的項目。如需提供使用者的 IAM 政策，請參閱 [安裝程式佈建資源的最低 IAM 政策](#)。如需如何擷取長期憑證的詳細資訊，請參閱《[IAM 使用者指南](#)》中的[管理 IAM 使用者的存取金鑰](#)。
 - b. 執行下列命令，將登入資料提供給 AWS IoT Greengrass 核心軟體。

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. (選用) 如果您建立了 IAM 使用者來佈建 Greengrass 裝置，請刪除該使用者。
- d. (選用) 如果您使用現有 IAM 使用者的存取金鑰 ID 和私密存取金鑰，請更新使用者的金鑰，使其不再有效。如需詳細資訊，請參閱《AWS Identity and Access Management 使用者指南》中的[更新存取金鑰](#)。

下載 AWS IoT Greengrass 核心軟體

您可以從下列位置下載最新版本的 AWS IoT Greengrass Core 軟體：

- <https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip>

Note

您可以從下列位置下載特定版本的 AWS IoT Greengrass Core 軟體。將##取代為要下載的版本。

```
https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
```


下載 AWS IoT Greengrass 核心軟體

1. 在您的核心裝置上，將 AWS IoT Greengrass Core 軟體下載至名為 `greengrass-nucleus-latest.zip` 的檔案。

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
```

下載此軟體，即表示您同意 [Greengrass 核心軟體授權合約](#) 之規定。

2. (選用) 驗證 Greengrass 核軟體簽章

Note

此功能適用於 Greengrass 核版本 2.9.5 及更新版本。

- a. 使用下列命令來驗證 Greengrass 核子成品的簽章：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。`jdk17.0.6_10` 將取代為您安裝的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

檔案名稱看起來可能會有所不同，視您安裝的 JDK 版本而定。*jdk17.0.6_10* 將取代為您安裝的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -  
verify -certs -verbose greengrass-nucleus-latest.zip
```

- b. jarsigner 調用會產生輸出，指出驗證的結果。
 - i. 如果已簽署 Greengrass 核 zip 檔案，輸出會包含下列陳述式：

```
jar verified.
```

- ii. 如果 Greengrass 核 zip 檔案未簽署，輸出會包含下列陳述式：

```
jar is unsigned.
```

- c. 如果您提供 Jarsigner -certs 選項以及 -verify 和 -verbose 選項，輸出也會包含詳細的簽署者憑證資訊。
3. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. (選用) 執行下列命令以查看 AWS IoT Greengrass 核心軟體版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

 Important

如果您安裝的 Greengrass 核版本早於 2.4.0 版，安裝 AWS IoT Greengrass 核心軟體後請勿移除此資料夾。Core AWS IoT Greengrass 軟體會使用此資料夾中的檔案來執行。

如果您下載最新版本的軟體，請安裝 v2.4.0 或更新版本，並在安裝 AWS IoT Greengrass Core 軟體後移除此資料夾。

安裝 AWS IoT Greengrass 核心軟體

使用指定 執行下列動作的引數執行安裝程式：

- 建立核心裝置操作所需的 AWS 資源。
- 指定 以使用ggc_user系統使用者在核心裝置上執行軟體元件。在 Linux 裝置上，此命令也會指定使用ggc_group系統群組，安裝程式會為您建立系統使用者和群組。
- 將 AWS IoT Greengrass Core 軟體設定為開機時執行的系統服務。在 Linux 裝置上，這需要 [Systemd](#) init 系統。

 Important

在 Windows 核心裝置上，您必須將 AWS IoT Greengrass Core 軟體設定為系統服務。

若要使用本機開發工具設定開發裝置，請指定 `--deploy-dev-tools true` 引數。安裝完成後，本機開發工具最多可能需要一分鐘的時間才能部署。

如需可指定引數的詳細資訊，請參閱 [安裝程式引數](#)。

Note

如果您在記憶體有限的 AWS IoT Greengrass 裝置上執行，您可以控制 AWS IoT Greengrass Core 軟體使用的記憶體數量。若要控制記憶體配置，您可以在核心元件的 `jvmOptions` 組態參數中設定 JVM 堆積大小選項。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。

安裝 AWS IoT Greengrass 核心軟體

1. 使用文字編輯器來建立名為 `config.yaml` 的組態檔案，以提供給安裝程式。

例如，在以 Linux 為基礎的系統上，您可以執行下列命令來使用 GNU nano 來建立 檔案。

```
nano GreengrassInstaller/config.yaml
```

將下列 YAML 內容複製到 檔案。此部分組態檔案會指定系統參數和 Greengrass 核參數。

```
---
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"
```

- 將 *us-west-2* 取代為您建立資源的 AWS 區域。
 - 將 *iotDataEndpoint* 取代為您的 AWS IoT 資料端點。
 - 將 *iotCredEndpoint* 取代為您的 AWS IoT 登入資料端點。
2. 執行 AWS IoT Greengrass Core 安裝程式。取代命令中的引數值，如下所示。

Note

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請使用根資料夾，例如 `C:\greengrass\v2` 或 `D:\greengrass\v2`，將 Greengrass 元件路徑保持在 260 個字元的限制以下。

- a. `/greengrass/v2` 或 `C#\greengrass\v2`：用來安裝 AWS IoT Greengrass Core 軟體的根資料夾路徑。
- b. `GreengrassInstaller`。解壓縮 AWS IoT Greengrass Core 軟體安裝程式的資料夾路徑。
- c. `##`。尋找或建立資源 AWS 區域的。
- d. `MyGreengrassCore`。Greengrass 核心裝置的 AWS IoT 物件名稱。如果物件不存在，安裝程式會建立它。安裝程式會下載憑證以驗證為 AWS IoT 實物。如需詳細資訊，請參閱[裝置身分驗證和授權 AWS IoT Greengrass](#)。

 Note

物件名稱不能包含冒號 (:) 字元。

- e. `MyGreengrassCoreGroup`。Greengrass 核心裝置的 AWS IoT 物件群組名稱。如果物件群組不存在，安裝程式會建立它，並將該物件新增至其中。如果物件群組存在且具有作用中部署，核心裝置會下載並執行部署指定的軟體。

 Note

物件群組名稱不能包含冒號 (:) 字元。

- f. `GreengrassV2IoTThingPolicy`。允許 Greengrass 核心裝置與 AWS IoT 和 通訊 AWS IoT 的政策名稱 AWS IoT Greengrass。如果 AWS IoT 政策不存在，安裝程式會使用此名稱建立寬鬆 AWS IoT 政策。您可以限制此政策的許可給您的使用案例。如需詳細資訊，請參閱[AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。
- g. `GreengrassV2TokenExchangeRole`。允許 Greengrass 核心裝置取得臨時 AWS 登入資料的 IAM 角色名稱。如果角色不存在，安裝程式會建立該角色，並建立和連接名為的政策 `GreengrassV2TokenExchangeRoleAccess`。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。
- h. `GreengrassCoreTokenExchangeRoleAlias`。IAM 角色的別名，允許 Greengrass 核心裝置稍後取得臨時登入資料。如果角色別名不存在，安裝程式會建立它，並將其指向您指定的 IAM 角色。如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

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

在 Windows 核心裝置上，您必須指定 `--setup-system-service true` 將 AWS IoT Greengrass Core 軟體設定為系統服務。

如果成功，安裝程式會列印下列訊息：

- 如果您指定 `--provision`，安裝程式會在成功設定資源 `Successfully configured Nucleus with provisioned resource details` 時列印。
 - 如果您指定 `--deploy-dev-tools`，安裝程式會在成功建立部署 `Configured Nucleus to deploy aws.greengrass.Cli component` 時列印。
 - 如果您指定 `--setup-system-service true`，安裝程式會在設定並執行軟體做為服務 `Successfully set up Nucleus as a system service` 時列印。
 - 如果您未指定 `--setup-system-service true`，安裝程式 `Launched Nucleus successfully` 會在成功時列印並執行軟體。
3. 如果您安裝 [v2Greengrass 核](#).0.4 或更新版本，請略過此步驟。如果您下載最新版本的軟體，則已安裝 v2.0.4 或更新版本。

執行下列命令，為您的 AWS IoT Greengrass Core 軟體根資料夾設定所需的檔案許可。`/greengrass/v2` 將取代為您安裝命令中指定的根資料夾，並將 `/greengrass` 取代為您根資料夾的父資料夾。

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

如果您將 AWS IoT Greengrass Core 軟體安裝為系統服務，安裝程式會為您執行軟體。否則，您必須手動執行軟體。如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)。

i Note

根據預設，安裝程式建立的 IAM 角色不允許存取 S3 儲存貯體中的元件成品。若要部署定義 Amazon S3 中成品的自訂元件，您必須將許可新增至角色，以允許核心裝置擷取元件成品。如需詳細資訊，請參閱[允許存取元件成品的 S3 儲存貯體](#)。

如果您還沒有元件成品的 S3 儲存貯體，您可以在建立儲存貯體之後新增這些許可。

如需如何設定和使用軟體的詳細資訊 AWS IoT Greengrass，請參閱以下內容：

- [設定 AWS IoT Greengrass 核心軟體](#)
- [開發 AWS IoT Greengrass 元件](#)
- [將 AWS IoT Greengrass 元件部署至裝置](#)
- [Greengrass 命令列界面](#)

FIPS 合規第一方元件

aws.greengrass.Nucleus	data.iot-fips. <i>us-east-1</i> .amazonaws.com
	greengrass-fips. <i>us-east-1</i> .amazonaws.com
	data.credentials.iot-fips. <i>us-east-1</i> .amazonaws.com
aws.greengrass.TokenExchangeService	data.credentials.iot-fips. <i>us-east-1</i> .amazonaws.com
aws.greengrass.Cli	
aws.greengrass.StreamManager	• kinesis-fips. <i>us-east-1</i> .amazonaws.com • data.iotsitewise-fips. <i>us-east-1</i> .amazonaws.com • s3-fips. <i>us-east-1</i> .amazonaws.com
	 Note 串流管理員不支援 AWS IoT Analytics FIPS 端點

<code>aws.greengrass.LogManager</code>	<code>log-fips.us-east-1.amazonaws.com</code>
<code>aws.greengrass.crypto.Pkcs11Provider</code>	
<code>aws.greengrass.ShadowManager</code>	
<code>aws.greengrass.DockerApplicationManager</code>	<code>ecr-fips.us-east-1.amazonaws.com</code>
<code>aws.greengrass.SecretManager</code>	<code>secretsmanager-fips.us-east-1.amazonaws.com</code>
<code>aws.greengrass.telemetry.NucleusEmitter</code>	
<code>aws.greengrass.clientdevices.IPDetector</code>	
<code>aws.greengrass.DiskSpooler</code>	

中的彈性 AWS IoT Greengrass

AWS 全球基礎設施是以 Amazon Web Services 區域和可用區域為基礎。每個 AWS 區域 提供多個實體隔離和隔離的可用區域，這些可用區域與低延遲、高輸送量和高度冗餘聯網連接。透過可用區域，您可以設計與操作的應用程式和資料庫，在可用區域之間自動容錯移轉而不會發生中斷。可用區域的可用性、容錯能力和擴展能力，均較單一或多個資料中心的傳統基礎設施還高。

如需詳細資訊，請參閱 [AWS 全球基礎設施](#)。

除了 AWS 全球基礎設施之外，AWS IoT Greengrass 還提供多種功能，以協助支援您的資料彈性和備份需求。

- 您可以設定 Greengrass 核心裝置，將日誌寫入本機檔案系統和 CloudWatch Logs。如果核心裝置失去連線，則可以繼續在檔案系統上記錄訊息。重新連線時，它會將日誌訊息寫入 CloudWatch Logs。如需詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。
- 如果核心裝置在部署期間斷電，則會在 AWS IoT Greengrass 核心軟體再次啟動後繼續部署。

- 如果核心裝置失去網際網路連線，Greengrass 用戶端裝置可以繼續透過本機網路進行通訊。
- 您可以撰寫 Greengrass 元件，以讀取[串流管理員](#)串流並將資料傳送至本機儲存目的地。

中的基礎設施安全性 AWS IoT Greengrass

作為受管服務，AWS IoT Greengrass 受到 [Amazon Web Services：安全程序概觀](#) 白皮書中所述的 AWS 全球網路安全程序的保護。

您可以使用 AWS 發佈的 API 呼叫，AWS IoT Greengrass 透過網路存取。用戶端必須支援 Transport Layer Security (TLS) 1.2 或更新版本。建議使用 TLS 1.3 或更新版本。用戶端也必須支援具備完美轉送私密 (PFS) 的密碼套件，例如臨時 Diffie-Hellman (DHE) 或橢圓曲線臨時 Diffie-Hellman (ECDHE)。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

請求必須使用存取金鑰 ID 和與 IAM 委託人相關聯的私密存取金鑰來簽署。或者，您可以使用 [AWS Security Token Service](#) (AWS STS) 產生臨時安全憑證來簽署請求。

在 AWS IoT Greengrass 環境中，裝置會使用 X.509 憑證和密碼編譯金鑰來連接和驗證 AWS 雲端。如需詳細資訊，請參閱[the section called “裝置身分驗證和授權”](#)。

中的組態和漏洞分析 AWS IoT Greengrass

IoT 環境可能由大量裝置組成，各自具有多樣化的功能、長時間在線上，並且散佈在多個地理位置。這些特點使裝置設定複雜且極易出錯。由於裝置通常受限於運算能力、記憶體和儲存功能，因此限制了裝置本身的加密和其他形式的安全性的使用。此外，裝置通常會使用具有已知漏洞的軟體。這些因素讓 IoT 裝置成為對駭客極具吸引力的目標，且難以持續保護裝置。

AWS IoT Device Defender 透過提供識別安全問題和偏離最佳實務的工具來解決這些挑戰。您可以使用 AWS IoT Device Defender 來分析、稽核和監控連線裝置，以偵測異常行為，並降低安全風險。AWS IoT Device Defender 可以稽核裝置，以確保其遵守安全最佳實務，並偵測裝置上的異常行為。這可讓您在裝置上強制執行一致的安全政策，並在裝置遭到入侵時快速回應。如需詳細資訊，請參閱下列主題：

- [Device Defender 元件](#)
- [AWS IoT Device Defender](#) (在 AWS IoT Core 開發人員指南中)。

在 AWS IoT Greengrass 環境中，您應該注意下列考量事項：

- 您須負責保護您的實體裝置、裝置上的檔案系統以及區域網路。

- AWS IoT Greengrass 不會對使用者定義的 Greengrass 元件強制執行網路隔離，無論它們是否在 Greengrass 容器中執行。因此，Greengrass 元件可以與系統或網路外部執行的任何其他程序進行通訊。

中的程式碼完整性 AWS IoT Greengrass V2

AWS IoT Greengrass 會將軟體元件從 部署 AWS 雲端 到執行 AWS IoT Greengrass Core 軟體的裝置。這些軟體元件包含 [AWS提供的元件](#)，以及您上傳至的 [自訂元件](#) AWS 帳戶。每個元件都由配方組成。配方會定義元件的中繼資料，以及任意數量的成品，這些成品是元件二進位檔，例如編譯的程式碼和靜態資源。元件成品存放在 Amazon S3 中。

當您開發和部署 Greengrass 元件時，請遵循這些基本步驟，在 AWS 帳戶 和 裝置上使用元件成品：

1. 建立成品並將其上傳至 S3 儲存貯體。
2. 從 AWS IoT Greengrass 服務中的配方和成品建立元件，這會計算每個成品的密碼[編譯雜湊](#)。
3. 將元件部署到 Greengrass 核心裝置，以下載和驗證每個成品的完整性。

AWS 負責在您將成品上傳至 S3 儲存貯體後維護成品的完整性，包括將元件部署至 Greengrass 核心裝置時。上傳成品到 S3 儲存貯體之前，您必須負責保護軟體成品的安全。您也必須負責保護對中資源的存取 AWS 帳戶，包括上傳元件成品的 S3 儲存貯體。

Note

Amazon S3 提供名為 S3 物件鎖定的功能，可用來防止 S3 儲存貯體中元件成品的變更 AWS 帳戶。您可以使用 S3 物件鎖定來防止刪除或覆寫元件成品。如需詳細資訊，請參閱《Amazon Simple Storage Service 使用者指南》中的[使用 S3 物件鎖定](#)。

AWS 發佈公有元件時，當您上傳自訂元件時，AWS IoT Greengrass 會計算每個元件成品的密碼編譯摘要。會 AWS IoT Greengrass 更新元件配方，以包含每個成品的摘要，以及用於計算該摘要的雜湊演算法。此摘要保證成品的完整性，因為如果成品在 AWS 雲端 中或在下載期間變更，其檔案摘要將與 AWS IoT Greengrass 存放在元件配方中的摘要不相符。如需詳細資訊，請參閱[元件配方參考中的成品](#)。

當您將元件部署到核心裝置時，AWS IoT Greengrass 核心軟體會下載元件配方和配方定義的每個元件成品。AWS IoT Greengrass 核心軟體會計算每個下載成品檔案的摘要，並將其與該成品在配方中的摘要進行比較。如果摘要不相符，部署會失敗，而 AWS IoT Greengrass Core 軟體會從裝置的檔案

系統刪除下載的成品。如需如何保護核心裝置與 AWS IoT Greengrass 之間連線的詳細資訊，請參閱[傳輸中加密](#)。

您有責任保護核心裝置檔案系統上元件成品檔案的安全。Core AWS IoT Greengrass 軟體會將成品儲存至 Greengrass 根 packages 資料夾中的 資料夾。您可以使用 AWS IoT Device Defender 來分析、稽核和監控核心裝置。如需詳細資訊，請參閱[中的組態和漏洞分析 AWS IoT Greengrass](#)。

AWS IoT Greengrass 和介面 VPC 端點 (AWS PrivateLink)

您可以建立介面 VPC 端點，在 VPC 與 AWS IoT Greengrass 控制平面之間建立私有連線。您可以使用此端點來管理 AWS IoT Greengrass 服務中的元件、部署和核心裝置。介面端點採用 [AWS PrivateLink](#) 技術，可讓您在沒有網際網路閘道、NAT 裝置、VPN 連接或 AWS Direct Connect 連接的情況下私下存取 AWS IoT Greengrass APIs。VPC 中的執行個體不需要公有 IP 地址即可與 AWS IoT Greengrass APIs 通訊。VPC 與 之間的流量 AWS IoT Greengrass 不會離開 Amazon 網路。

每個介面端點都是由您子網路中的一或多個[彈性網路介面](#)表示。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[介面 VPC 端點 \(AWS PrivateLink\)](#)。

主題

- [AWS IoT Greengrass VPC 端點的考量事項](#)
- [建立 AWS IoT Greengrass 控制平面操作的介面 VPC 端點](#)
- [為 建立 VPC 端點政策 AWS IoT Greengrass](#)
- [在 VPC 中操作 AWS IoT Greengrass 核心裝置](#)

AWS IoT Greengrass VPC 端點的考量事項

設定介面 VPC 端點之前 AWS IoT Greengrass，請檢閱《Amazon VPC 使用者指南》中的[介面端點屬性和限制](#)。此外，請注意下列考量事項：

- AWS IoT Greengrass 支援從您的 VPC 呼叫其所有控制平面 API 動作。控制平面包含 [CreateDeployment](#) 和 [ListEffectiveDeployments](#) 等操作。控制平面不包含 [ResolveComponentCandidates](#) 和 [Discover](#) 等操作，這是資料平面操作。
- AWS 中國區域 AWS IoT Greengrass 目前不支援的 VPC 端點。

建立 AWS IoT Greengrass 控制平面操作的介面 VPC 端點

您可以使用 Amazon VPC 主控台或 AWS Command Line Interface () 為 AWS IoT Greengrass 控制平面建立 VPC 端點AWS CLI。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[建立介面端點](#)。

AWS IoT Greengrass 使用下列服務名稱建立的 VPC 端點：

- `com.amazonaws.region.greengrass`

如果您為端點啟用私有 DNS，則可以 AWS IoT Greengrass 使用區域的預設 DNS 名稱向 提出 API 請求，例如 `greengrass.us-east-1.amazonaws.com`。預設為啟用私有 DNS。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[透過介面端點存取服務](#)。

為 建立 VPC 端點政策 AWS IoT Greengrass

您可以將端點政策連接至 VPC 端點，以控制控制平面操作的 AWS IoT Greengrass 存取。此政策會指定下列資訊：

- 可執行動作的主體。
- 委託人可以執行的動作。
- 委託人可以對其執行動作的資源。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的[使用 VPC 端點控制對服務的存取](#)。

Example範例：AWS IoT Greengrass 動作的 VPC 端點政策

以下是 端點政策的範例 AWS IoT Greengrass。連接到端點時，此政策會授予所有資源上所有委託人的所列 AWS IoT Greengrass 動作的存取權。

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "greengrass:CreateDeployment",
        "greengrass:ListEffectiveDeployments"
      ],
      "Resource": "*"
    }
  ]
}
```

```
}  
]  
}
```

在 VPC 中操作 AWS IoT Greengrass 核心裝置

您可以操作 Greengrass 核心裝置，並在 VPC 中執行部署，而無需公有網際網路存取。您至少必須使用對應的 DNS 別名設定下列 VPC 端點。如需如何建立和使用 VPC 端點的詳細資訊，請參閱《Amazon [VPC 使用者指南](#)》中的[建立 VPC 端點](#)。

Note

AWS IoT data 和 AWS IoT 登入資料的自動建立 DNS 記錄的 VPC 功能已停用。若要連接這些端點，您必須手動建立私有 DNS 記錄。如需詳細資訊，請參閱[介面端點的私有 DNS](#)。如需 AWS IoT Core VPC 限制的詳細資訊，請參閱[VPC 端點的限制](#)。

先決條件

- 您必須使用手動佈建步驟安裝 AWS IoT Greengrass 核心軟體。如需詳細資訊，請參閱[使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。

限制

- 中國區域 和 不支援在 VPC 中操作 Greengrass 核心裝置 AWS GovCloud (US) Regions。
- 如需 AWS IoT data 和 AWS IoT 登入資料提供者 VPC 端點限制的詳細資訊，請參閱[限制](#)。

設定您的 Greengrass 核心裝置以在 VPC 中操作

1. 取得的 AWS IoT 端點 AWS 帳戶，並儲存它們以供稍後使用。您的裝置會使用這些端點來連線 AWS IoT。請執行下列操作：
 - a. 取得 AWS IoT 的資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

如果請求成功，回應看起來與下列範例類似。


```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

- b. 取得的 AWS IoT 登入資料端點 AWS 帳戶。

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

如果請求成功，回應看起來與下列範例類似。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

2. 為 AWS IoT data 和 AWS IoT 登入資料端點建立 Amazon VPC 界面：

- a. 導覽至 [VPC](#) 端點主控台，在左側選單的虛擬私有雲端下，選擇端點，然後選擇建立端點。
- b. 在建立端點頁面中，指定下列資訊。
- 選擇服務類別的 AWS 服務。
 - 針對 Service Name (服務名稱)，輸入關鍵字 `iot` 進行搜尋。在顯示的 `iot` 服務清單中，選擇端點。

如果您為 AWS IoT Core 資料平面建立 VPC 端點，請選擇 AWS IoT Core 您區域的資料平面 API 端點。端點的格式為 `com.amazonaws.region.iot.data`。

如果您為 AWS IoT Core 登入資料提供者建立 VPC 端點，請選擇您區域的 AWS IoT Core 登入資料提供者端點。端點的格式為 `com.amazonaws.region.iot.credentials`。

 Note

中國區域中 AWS IoT Core 資料平面的服務名稱格式為 `cn.com.amazonaws.region.iot.data`。中國區域不支援為 AWS IoT Core 登入資料提供者建立 VPC 端點。

- 針對 VPC 和 Subnets (子網路)，選擇要在其中建立端點的 VPC，以及要在其中建立端點網路的可用區域 (AZ)。

- 針對 Enable DNS name (啟用 DNS 名稱)，確定未選取 Enable for this endpoint (為此端點啟用)。AWS IoT Core 資料平面和 AWS IoT Core 登入資料提供者都不支援私有 DNS 名稱。
 - 針對 Security group (安全群組)，選擇要與端點網路介面建立關聯的安全群組。
 - 您可以選擇性地新增或移除標籤。標籤是您用來與端點建立關聯的名稱值對。
- c. 若要建立 VPC 端點，請選擇 Create endpoint (建立端點)。
3. 建立 AWS PrivateLink 端點之後，您會在端點的詳細資訊索引標籤中看到 DNS 名稱清單。您可以使用您在本節中建立的其中一個 DNS 名稱來[設定私有託管區域](#)。
 4. 建立 Amazon S3 端點。如需詳細資訊，請參閱[建立 Amazon S3 的 VPC 端點](#)。
 5. 如果您使用的是 [AWS 提供的 Greengrass 元件](#)，則可能需要額外的端點和組態。若要檢視端點需求，請從 AWS 提供的元件清單中選取元件，然後查看需求區段。例如，[日誌管理員元件需求](#)建議此元件必須能夠對端點執行傳出請求 `logs.region.amazonaws.com`。
- 如果您使用自己的元件，您可能需要檢閱相依性並執行其他測試，以判斷是否需要任何其他端點。
6. 在 Greengrass 核組態中，`greengrassDataPlaneEndpoint` 必須設定為 `iotdata`。如需詳細資訊，請參閱 [Greengrass 核組態](#)。
 7. 如果您位於 `us-east-1` 區域，請在 Greengrass 核組態 **REGIONAL** 中將組態參數 `s3EndpointType` 設定為 `REGIONAL`。此功能適用於 Greengrass 核版本 2.11.3 或更新版本。

Example 範例：元件組態

```
{
  "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"
      ...
    }
  }
}
```

下表提供有關對應自訂私有 DNS 別名的資訊。

服務	VPC 端點服務名稱	VPC 端點類型	自訂私有 DNS 別名	備註
AWS IoT data	com.amazonaws. <i>region</i> .iot.data	介面	<i>prefix</i> -ats.iot. <i>region</i> .s.com	私有 DNS 記錄應符合您帳戶的 AWS IoT data 端點：aws-iot-describe-endpoint--endpoint-type-iot:Data-ATS。
AWS IoT 登入資料	com.amazonaws. <i>region</i> .iot.credentials	介面	<i>prefix</i> .s.com	私有 DNS 記錄應與您的帳戶 AWS IoT 登入資料端點相符：aws-iot-describe-endpoint--endpoint-type-iot:Credentials

服務	VPC 端點服務名稱	VPC 端點類型	自訂私有 DNS 別名	備註
				ntialProvider。
Amazon S3	com.amazonaws. <i>region</i> .s3	介面		DNS 記錄會自動建立。

的安全最佳實務 AWS IoT Greengrass

本主題包含 的安全最佳實務 AWS IoT Greengrass。

盡可能授予最低的許可

以無權限使用者身分執行元件的最低權限原則。除非絕對必要，否則元件不應以根執行。

在 IAM 角色中使用一組最低許可。限制對 IAM 政策中的 Action 和 Resource 屬性使用 * 萬用字元。相反地，在可能的情況下，宣告一組有限的動作和資源。如需最低權限和其他原則最佳實務的詳細資訊，請參閱 [the section called “政策最佳實務”](#)。

最低權限最佳實務也適用於您連接至 Greengrass 核心 AWS IoT 的政策。

請勿在 Greengrass 元件中硬式編碼登入資料

請勿在使用者定義的 Greengrass 元件中硬式編碼登入資料。為了更妥善地保護您的登入資料：

- 若要與服務互動 AWS，請在 [Greengrass 核心裝置服務角色](#) 中定義特定動作和資源的許可。
- 使用 [秘密管理員元件](#) 來存放您的登入資料。或者，如果函數使用 AWS SDK，請使用預設登入資料提供者鏈結中的登入資料。

請勿記錄敏感資訊

您應該防止記錄登入資料和其他個人識別資訊 (PII)。即使存取核心裝置上的本機日誌需要根權限，而且存取 CloudWatch Logs 需要 IAM 許可，仍建議您實作下列保護措施。

- 請勿在 MQTT 主題路徑中使用敏感資訊。
- 請勿在 AWS IoT Core 登錄中的裝置 (物件) 名稱、類型和屬性中使用敏感資訊。
- 請勿在使用者定義的 Greengrass 元件或 Lambda 函數中記錄敏感資訊。
- 請勿在 Greengrass 資源的名稱和 ID 中使用敏感資訊：
 - 核心裝置
 - 元件
 - 部署
 - Loggers

讓裝置的時鐘保持同步

在裝置上保持準確的時間是很重要的。X.509 憑證具到期日期和時間。裝置上的時鐘用來驗證伺服器憑證是否仍然有效。裝置時鐘可能會隨著時間而偏移，或是電池可能會放電。

如需詳細資訊，請參閱《[開發人員指南](#)》中的[保持裝置的時鐘同步](#)最佳實務。AWS IoT Core

密碼套件建議

Greengrass 預設會選取裝置上可用的最新 TLS 密碼套件。考慮在裝置上停用舊版密碼套件。例如，CBC 密碼套件。

如需詳細資訊，請參閱 [Java 密碼編譯組態](#)。

另請參閱

- 《AWS IoT 開發人員指南》中的[安全最佳實務 AWS IoT Core](#)
- AWS 物聯網官方部落格上[工業 IoT 解決方案的十項安全黃金規則](#)

使用 AWS IoT Device Tester for AWS IoT Greengrass V2

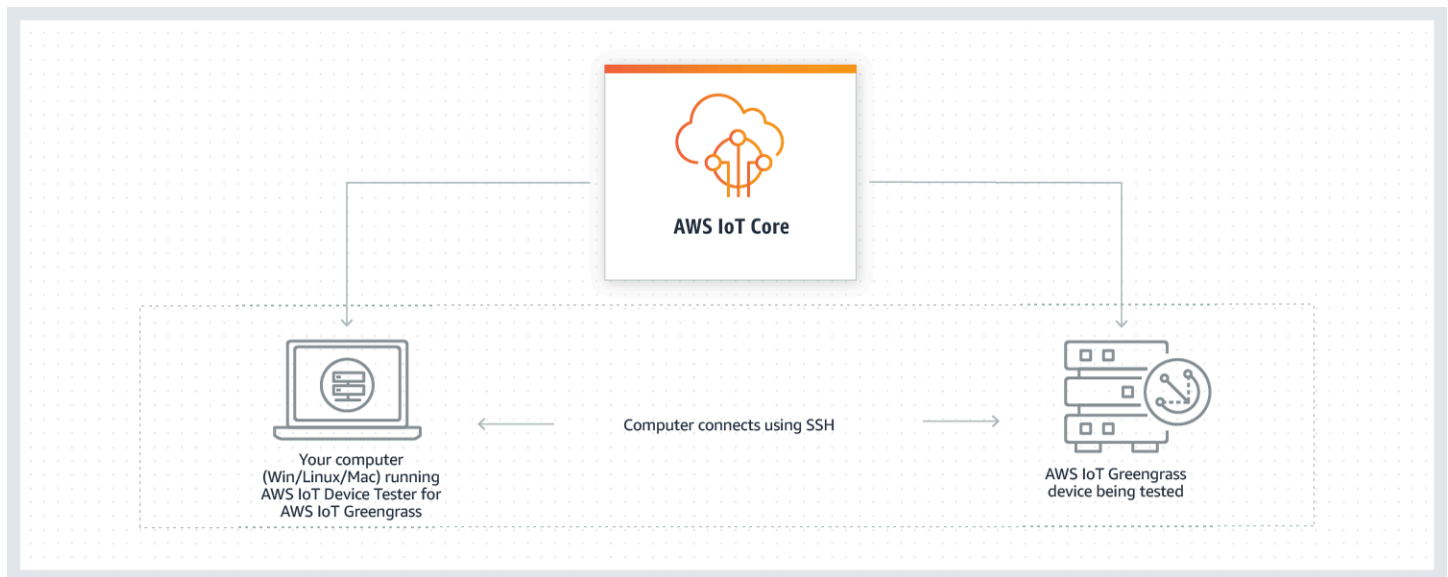
AWS IoT Device Tester (IDT) 是可下載的測試架構，可讓您驗證 IoT 裝置。您可以使用 IDT for AWS IoT Greengrass 來執行資格套件，並為您的裝置建立和執行自訂測試套件。

IDT for 會在連接到要測試之裝置的主機電腦 (Windows、macOS 或 Linux) AWS IoT Greengrass 上執行。它會執行測試並彙總結果。它還提供命令列界面來管理測試程序。

AWS IoT Greengrass 資格套件

使用 AWS IoT Device Tester for AWS IoT Greengrass V2 來驗證 AWS IoT Greengrass 核心軟體是否在您的硬體上執行，並且可以與 通訊 AWS 雲端。它也會使用 end-to-end 測試 AWS IoT Core。例如，它會驗證您的裝置是否可以部署元件並進行升級。

如果您想要將硬體新增至 AWS Partner Device Catalog，請 AWS IoT Greengrass 執行資格套件來產生可提交的測試報告 AWS IoT。如需詳細資訊，請參閱 [AWS 裝置資格計劃](#)。



IDT for AWS IoT Greengrass V2 使用測試套件和測試群組的概念來組織測試。

- 測試套件是一組測試群組，用來驗證裝置是否適用於特定版本的 AWS IoT Greengrass。
- 測試群組是與特定功能相關的一組個別測試，例如元件部署。

如需詳細資訊，請參閱[使用 IDT AWS IoT Greengrass 執行資格套件](#)。

自訂測試套件

從 IDT v4.0.1 開始，IDT for AWS IoT Greengrass V2 結合了標準化的組態設定和結果格式，以及測試套件環境，可讓您為裝置和裝置軟體開發自訂測試套件。您可以為自己的內部驗證新增自訂測試，或將其提供給客戶進行裝置驗證。

測試寫入器如何設定自訂測試套件，決定執行自訂測試套件所需的設定組態。如需詳細資訊，請參閱[使用 IDT 來開發和執行您自己的測試套件](#)。

for AWS IoT Greengrass V2 AWS IoT Device Tester 支援的版本

本主題列出支援的 IDT for AWS IoT Greengrass V2 版本。最佳實務是建議您使用支援目標版本 AWS IoT Greengrass V2 的最新版本 IDT for AWS IoT Greengrass V2。的新版本 AWS IoT Greengrass 可能需要您下載新版本的 IDT for AWS IoT Greengrass V2。如果 IDT for AWS IoT Greengrass V2 與您 AWS IoT Greengrass 正在使用的 版本不相容，您會在啟動測試執行時收到通知。

下載軟體即表示您同意[AWS IoT Device Tester 授權合約](#)。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

最新版 IDT for AWS IoT Greengrass V2

您可以使用此版本的 IDT for AWS IoT Greengrass V2 搭配此處列出的 AWS IoT Greengrass 版本。

適用於 的 IDT v4.9.4 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- [Greengrass 核](#) v2.12.0、v2.11.0、v2.10.0 和 v2.9.5

IDT 軟體下載：

- IDT v4.9.4 搭配測試套件 GGV2Q_2.5.4 for [Linux](#)
- IDT v4.9.4 搭配適用於 [macOS](#) 的測試套件 GGV2Q_2.5.4
- IDT v4.9.4 搭配測試套件 GGV2Q_2.5.4 for [Windows](#)

版本備註：

- 針對執行 AWS IoT Greengrass Core 軟體版本 2.12.0、2.11.0、2.10.0 和 2.9.5 的裝置，啟用裝置驗證和資格。
- 移除串流管理員和機器學習測試群組。

其他備註：

- 如果您的裝置使用 HSM，且您使用的是 2.10.x 核心，請遷移至 Greengrass 2.11.0 版或更新版本。

測試套件版本：

GGV2Q_2.5.4

- 版 2024.05.03

的早期 IDT 版本 AWS IoT Greengrass

也支援下列舊版的 IDT for AWS IoT Greengrass V2。

適用於 的 IDT v4.9.3 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- [Greengrass 核](#) v2.12.0、v2.11.0、v2.10.0 和 v2.9.5

IDT 軟體下載：

- IDT v4.9.3 搭配測試套件 GGV2Q_2.5.3 for [Linux](#)
- 適用於 [macOS](#) 的測試套件 GGV2Q_2.5.3 的 IDT v4.9.3
- IDT v4.9.3 搭配測試套件 GGV2Q_2.5.3 for [Windows](#)

版本備註：

- 修正從 Windows 主機測試 Linux 裝置時元件測試中的問題，反之亦然。
- 從localcomponent測試群組中移除component測試案例。資格不再需要此測試案例。

其他備註：

- 如果您的裝置使用 HSM，且您使用的是 2.10.x 核心，請遷移至 Greengrass 2.11.0 版或更新版本。

測試套件版本：

GGV2Q_2.5.3

- 版 2024.04.05

不支援的 AWS IoT Device Tester for AWS IoT Greengrass V2 版本

本主題列出不支援的 IDT for AWS IoT Greengrass V2 版本。不支援的版本不會收到錯誤修正或更新。如需詳細資訊，請參閱[the section called “適用於 AWS IoT Device Tester 的支援政策 AWS IoT Greengrass”](#)。

適用於的 IDT v4.9.2 AWS IoT Greengrass

版本備註：

- 修正 Lambda 測試套件因 Java 8 已棄用而失敗的問題。

測試套件版本：

GGV2Q_2.5.2

- 版 2024.03.18

適用於的 IDT v4.9.1 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體版本 2.12.0、2.11.0、2.10.0 和 2.9.5 的裝置。
- 次要錯誤修正。

測試套件版本：

GGV2Q_2.5.1

- 版 2023.10.05

適用於的 IDT v4.7.0 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- [Greengrass 核 v2.11.0](#)、v2.10.0 和 v2.9.5

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體版本 2.11.0、2.10.0 和 2.9.5 的裝置。
- 新增支援，以在 AWS Systems Manager 參數存放區中存放 IDT 使用者資料值，並使用預留位置語法將其擷取到組態中。
- 次要錯誤修正。

測試套件版本：

GGV2Q_2.5.0

- 版 2022.12.13

適用於的 IDT v4.5.11 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體版本 2.9.1、2.9.0、2.8.1、2.8.0、2.7.0 和 2.6.0 的裝置。
- 新增在核心裝置上測試 PreInstalled Greengrass 的支援。
- 次要錯誤修正。

測試套件版本：

GGV2Q_2.4.1

- 版 2022.10.13

適用於的 IDT v4.5.8 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體版本 2.7.0、2.6.0 和 2.5.6 的裝置。
- 可讓您在核心裝置上使用 PreInstalled Greengrass 進行測試。
- 次要錯誤修正。

測試套件版本：

GGV2Q_2.4.0

- 版 2022.08.12

適用於的 IDT v4.5.3 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體版本 2.7.0、2.6.0、2.5.6、2.5.5、2.5.4 和 2.5.3 的裝置。
- 更新 DockerApplicationManager 測試以使用 ECR 型 docker 映像。
- 次要錯誤修正。

測試套件版本：

GGV2Q_2.3.1

- 版 2022.04.15

適用於的 IDT v4.5.1 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體 v2.5.3 的裝置。

- 新增支援，以驗證和驗證使用硬體安全模組 (HSM) 來存放 AWS IoT Greengrass Core 軟體所使用的私有金鑰和憑證的 Linux 型裝置。
- 實作新的 IDT 測試協調器來設定自訂測試套件。如需詳細資訊，請參閱[設定 IDT 測試協調器](#)。
- 其他次要錯誤修正。

測試套件版本：

GGV2Q_2.3.0

- 版 2022.01.11

適用於的 IDT v4.4.1 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體 v2.5.2 的裝置。
- 新增使用使用者定義 IAM 角色做為權杖交換角色的支援，待測裝置會擔任此角色來與 AWS 資源互動。

您可以在 [userdata.json](#) 檔案中指定 IAM 角色。如果您指定自訂角色，IDT 會使用該角色，而不是在測試執行期間建立預設字符交換角色。

- 其他次要錯誤修正。

測試套件版本：

GGV2Q_2.2.1

- 版 2021.12.12

適用於的 IDT v4.4.0 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體 v2.5.0 的裝置。
- 新增在 Windows 上執行 AWS IoT Greengrass Core 軟體的驗證和合格裝置的支援。
- 支援使用公有金鑰驗證進行安全 shell (SSH) 裝置連線。
- 使用安全最佳實務改善 IDT 許可 IAM 政策。
- 其他次要錯誤修正。

測試套件版本：

GGV2Q_2.1.0

- 版 2021.11.19

適用於的 IDT v4.2.0 AWS IoT Greengrass

版本備註：

- 包括對執行 AWS IoT Greengrass Core 軟體 v2.2.0 及更新版本之裝置上的下列功能資格的支援：
 - Docker—驗證裝置可以從 Amazon Elastic Container Registry (Amazon ECR) 下載 Docker 容器映像。
 - 機器學習 - 驗證裝置可以使用[深度學習執行期](#)或 [TensorFlow Lite](#) ML 架構執行機器學習 (ML) 推論。
 - 串流管理員 - 驗證裝置可以下載、安裝和執行 AWS IoT Greengrass 串流管理員。
- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體 v2.4.0、v2.3.0、v2.2.0 和 v2.1.0 的裝置。
- 將每個測試案例的測試日誌分組到 `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` 目錄中的個別 `<test-case-id>` 資料夾。
- 其他次要錯誤修正。

測試套件版本：

GGV2Q_2.0.1

- 版 2021.08.31

適用於的 IDT v4.1.0 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體 v2.3.0、v2.2.0、v2.1.0 和 v2.0.5 的裝置。
- 透過移除指定 GreengrassNucleusVersion 和 GreengrassCLIVersion 屬性的需求來改善 userdata.json 組態。
- 包括對 AWS IoT Greengrass Core 軟體 v2.1.0 和更新版本 Lambda 和 MQTT 功能資格的支援。您現在可以使用 IDT for AWS IoT Greengrass V2 來驗證您的核心裝置是否可以執行 Lambda 函數，以及裝置是否可以發佈和訂閱 AWS IoT Core MQTT 主題。
- 改善記錄功能。
- 其他次要錯誤修正。

測試套件版本：

GGV2Q_1.1.1

- 版 2021.06.18

適用於的 IDT v4.0.2 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行 AWS IoT Greengrass Core 軟體 v2.1.0 的裝置。
- 新增對 AWS IoT Greengrass Core 軟體 v2.1.0 和更新版本 Lambda 和 MQTT 功能資格的支援。您現在可以使用 IDT for AWS IoT Greengrass V2 來驗證您的核心裝置是否可以執行 Lambda 函數，以及裝置是否可以發佈和訂閱 AWS IoT Core MQTT 主題。
- 改善記錄功能。
- 其他次要錯誤修正。

測試套件版本：

GGV2Q_1.1.1

- 版 2021.05.05

適用於的 IDT v4.0.1 AWS IoT Greengrass

版本備註：

- 可讓您驗證並限定執行第 2 AWS IoT Greengrass 版軟體的裝置。
- 可讓您使用 AWS IoT Device Tester for 開發和執行自訂測試套件 AWS IoT Greengrass。如需詳細資訊，請參閱[使用 IDT 來開發和執行您自己的測試套件](#)。
- 提供 macOS 和 Windows 的程式碼簽署 IDT 應用程式。在 macOS 上，您可能需要授予 IDT 的安全例外狀況。如需詳細資訊，請參閱[macOS 上的安全例外狀況](#)。

測試套件版本：

GGV2Q_1.0.0

- 版 2020.12.22
- 除非您將features陣列value中對應的 設定為 ，否則測試套件只會執行資格所需的測試yes。

下載 IDT for AWS IoT Greengrass V2

本主題說明下載 AWS IoT Device Tester for AWS IoT Greengrass V2 的選項。您可以使用下列其中一個軟體下載連結，也可以依照指示以程式設計方式下載 IDT。

主題

- [手動下載 IDT](#)
- [以程式設計方式下載 IDT](#)

下載軟體即表示您同意[AWS IoT Device Tester 授權合約](#)。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

手動下載 IDT

本主題列出支援的 IDT for AWS IoT Greengrass V2 版本。最佳實務是，建議您使用支援目標版本 AWS IoT Greengrass V2 的最新版本 IDT for AWS IoT Greengrass V2。的新版本 AWS IoT Greengrass 可能需要您下載新版本的 IDT for AWS IoT Greengrass V2。如果 IDT for AWS IoT Greengrass V2 與您正在使用的 AWS IoT Greengrass 版本不相容，您會在啟動測試執行時收到通知。

適用於的 IDT v4.9.4 AWS IoT Greengrass

支援的 AWS IoT Greengrass 版本：

- [Greengrass 核](#) v2.12.0、v2.11.0、v2.10.0 和 v2.9.5

IDT 軟體下載：

- IDT v4.9.4 搭配 [Linux](#) 版測試套件 GGV2Q_2.5.4
- IDT v4.9.4 搭配 [macOS](#) 的測試套件 GGV2Q_2.5.4
- IDT v4.9.4 搭配測試套件 GGV2Q_2.5.4 for [Windows](#)

版本備註：

- 為執行 AWS IoT Greengrass Core 軟體 2.12.0、2.11.0、2.10.0 和 2.9.5 版的裝置啟用裝置驗證和資格驗證。
- 移除串流管理員和機器學習測試群組。

其他備註：

- 如果您的裝置使用 HSM，且您使用 2.10.x 的核，請遷移至 Greengrass 核 2.11.0 版或更新版本。

測試套件版本：

GGV2Q_2.5.4

- 2024.05.03 版

以程式設計方式下載 IDT

IDT 提供 API 操作，可用來擷取 URL，以程式設計方式下載 IDT。您也可以使用此 API 操作來檢查您是否有最新版本的 IDT。此 API 操作具有下列端點。

```
https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt
```

若要呼叫此 API 操作，您必須具有執行 **iot-device-tester:LatestIdt** 動作的許可。包含您的 AWS 簽章，並使用 **iot-device-tester** 做為服務名稱。

API 請求

HostOs主機機器的作業系統。您可以從以下選項中選擇：

- mac
- linux
- windows

TestSuiteType – 測試套件的類型。選擇下列選項：

GGV2 – IDT for AWS IoT Greengrass V2

ProductVersion

(選用) Greengrass 核的版本。服務會針對該版本的 Greengrass 核傳回最新的相容 IDT 版本。如果您未指定此選項，服務會傳回最新版本的 IDT。

API 回應

API 回應的格式如下。DownloadURL 包含 zip 檔案。

```
{
  "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
  }
}
```

範例

您可以參考下列範例以程式設計方式下載 IDT。這些範例使用您存放在 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 環境變數中的登入資料。若要遵循最佳安全實務，請勿將登入資料存放在程式碼中。

Example 範例：使用 cURL 7.75.0 版或更新版本 (Mac 和 Linux) 下載

如果您有 cURL 7.75.0 版或更新版本，您可以使用 `aws-sigv4` 旗標簽署 API 請求。此範例使用 [jq](#) 從回應剖析下載 URL。

Warning

`aws-sigv4` 旗標需要 curl GET 請求的查詢參數依 `HostOs/ProductVersion/TestSuiteType` 或的順序排列 `HostOs/TestSuiteType`。不符合的順序會導致從 API Gateway 取得正式字串不相符簽章時發生錯誤。

如果 `ProductVersion` 包含選用參數，您必須使用支援的產品版本，如 [AWS IoT Device Tester for AWS IoT Greengrass V2 的支援版本](#) 所述。

- 將 `us-west-2` 取代為您的 AWS 區域。如需區域代碼清單，請參閱 [區域端點](#)。
- 將 `linux` 取代為主機機器的作業系統。
- 將 `2.5.3 ##` 為您的 核版本 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 範例：使用舊版 cURL (Mac 和 Linux) 下載

您可以使用下列 cURL 命令搭配您簽署和計算的 AWS 簽章。如需如何簽署和計算 AWS 簽章的詳細資訊，請參閱 [簽署 AWS API 請求](#)。

- 將 `linux` 取代為主機機器的作業系統。

- 將####取代為日期和時間，例如 **20220210T004606Z**。
- 將##取代為日期，例如 **20220210**。
- 將 **AWSRegion** 取代為 AWS 區域。如需區域代碼清單，請參閱[區域端點](#)。
- 將 **AWSSignature** 取代為您產生的[AWS 簽章](#)。

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
Host0s=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範例：使用 Python 指令碼下載

此範例使用 Python [請求](#) 程式庫。此範例改編自 Python 範例，以在 AWS 一般參考中[簽署 AWS API 請求](#)。

- 將 **us-west-2** 取代為您的區域。如需區域代碼清單，請參閱[區域端點](#)。
- 將 **linux** 取代為主機機器的作業系統。

```
# 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)
```

使用 IDT AWS IoT Greengrass 執行資格套件

您可以使用 AWS IoT Device Tester for AWS IoT Greengrass V2 驗證 AWS IoT Greengrass Core 軟體是否在您的硬體上執行，並且可以與通訊 AWS 雲端。它也會使用 end-to-end 測試 AWS IoT Core。例如，它會驗證您的裝置是否可以部署元件並進行升級。

除了測試裝置之外，IDT for AWS IoT Greengrass V2 還會在您的 中建立資源（例如 AWS IoT 物件、群組等）AWS 帳戶，以促進資格程序。

為了建立這些資源，IDT for AWS IoT Greengrass V2 會使用 config.json 檔案中設定的 AWS 登入資料，代表您進行 API 呼叫。系統會在測試期間的不同時間點內佈建這些資源。

當您使用 IDT for AWS IoT Greengrass V2 AWS IoT Greengrass 執行資格套件時，它會執行下列步驟：

1. 載入並驗證您的裝置和登入資料組態。
2. 對所需的本機和雲端資源執行選取的測試。
3. 清除本機和雲端資源。
4. 產生測試報告以指出主機板是否通過符合資格所需的測試。

測試套件版本

IDT for AWS IoT Greengrass V2 會將測試組織成測試套件和測試群組。

- 測試套件是一組測試群組，用來驗證裝置是否適用於特定版本的 AWS IoT Greengrass。
- 測試群組是一組與特定功能相關的個別測試，例如元件部署。

測試套件使用 *major.minor.patch* 格式進行版本控制，例如 GGV2Q_1.0.0。當您下載 IDT 時，套件包含最新的 Greengrass 資格套件版本。

Important

來自不支援的測試套件版本的測試不符合裝置資格。IDT 不會列印不支援版本的資格報告。如需詳細資訊，請參閱[the section called “適用於 AWS IoT Device Tester 的支援政策 AWS IoT Greengrass”](#)。

您可以執行 `list-supported-products` 來列出目前 IDT 版本支援的 AWS IoT Greengrass 和 測試套件版本。

測試群組描述

核心資格的必要測試群組

需要這些測試群組，才能讓您的 AWS IoT Greengrass V2 裝置符合 AWS Partner Device Catalog 的資格。

核心相依性

驗證裝置是否符合 AWS IoT Greengrass Core 軟體的所有軟體和硬體需求。此測試群組包含下列測試案例：

Java 版本

檢查測試下的裝置上是否安裝了所需的 Java 版本。AWS IoT Greengrass 需要 Java 8 或更新版本。

PreTest驗證

檢查裝置是否符合執行測試的軟體需求。

- 對於以 Linux 為基礎的裝置，此測試會檢查裝置是否可以執行下列 Linux 命令：

`chmod, cp, echo, grep, kill, ln, mkinfo, ps, rm, sh, uname`

- 對於 Windows 型裝置，此測試會檢查裝置是否已安裝下列 Microsoft 軟體：

[Powershell](#) v5.1 或更新版本、[.NET](#) v4.6.1 或更新版本、[Visual C++](#) 2017 或更新版本、[PsExec 公用程式](#)

版本檢查程式

檢查 AWS IoT Greengrass 提供的 版本是否與您正在使用的 AWS IoT Device Tester 版本相容。

元件

驗證裝置是否可以部署元件並進行升級。此測試群組包含下列測試：

雲端元件

驗證雲端元件的裝置功能。

本機元件

驗證本機元件的裝置功能。

Lambda

此測試不適用於 Windows 型裝置。

驗證裝置是否可以部署使用 Java 執行時間的 Lambda 函數元件，以及 Lambda 函數可以使用 AWS IoT Core MQTT 主題做為工作訊息的事件來源。

MQTT

驗證裝置是否可以訂閱和發佈至 AWS IoT Core MQTT 主題。

選用測試群組

Note

這些測試群組是選用的，僅用於合格的 Linux Greengrass 核心裝置。如果您選擇符合選用測試的資格，您的裝置會在 AWS Partner Device Catalog 中列出其他功能。

Docker 相依性

驗證裝置是否符合所有必要的技術相依性，以使用 AWS 提供的 Docker 應用程式管理員 (aws.greengrass.DockerApplicationManager) 元件。

Docker Application Manager 資格

驗證裝置可以從 Amazon ECR 下載 Docker 容器映像。

Machine Learning相依性

Note

只有 IDT v4.9.3 支援機器學習選用測試群組。

驗證裝置是否符合使用 AWS 提供的機器學習 (ML) 元件所需的所有必要技術相依性。

Machine Learning推論測試

Note

只有 IDT v4.9.3 支援機器學習選用測試群組。

驗證裝置是否可以使用[深度學習執行期](#)和 [TensorFlow Lite](#) ML 架構執行 ML 推論。

串流管理員相依性

Note

僅在 IDT v4.9.3 中支援串流管理員選用測試群組。

驗證裝置是否可以下載、安裝和執行[AWS IoT Greengrass 串流管理員](#)。

硬體安全整合 (HSI)

Note

此測試僅適用於 Linux 型裝置的 IDT v4.9.3 和更新版本。目前 AWS IoT Greengrass 不支援 Windows 裝置的硬體安全整合。

驗證裝置是否可以使用存放在硬體安全模組 (HSM) 中的私有金鑰和憑證來驗證 AWS IoT 和 AWS IoT Greengrass 服務的連線。此測試也會驗證 AWS 提供的 [PKCS#11 供應商元件](#) 是否可以使用廠商提供的 PKCS#11 程式庫與 HSM 連接。如需詳細資訊，請參閱[硬體安全整合](#)。

執行 AWS IoT Greengrass 資格套件的先決條件

本節說明使用 AWS IoT Device Tester (IDT) 的先決條件 AWS IoT Greengrass。

下載 AWS IoT Device Tester 適用於 的最新版本 AWS IoT Greengrass

下載[最新版本](#)的 IDT，並將軟體解壓縮到檔案系統上具有讀取/寫入許可的位置 (*<device-tester-extract-location>*)。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請將 IDT 解壓縮到根目錄，例如 C:\ 或 D:\，使路徑保持在 260 個字元的限制以下。

下載 AWS IoT Greengrass 軟體

IDT for AWS IoT Greengrass V2 會測試您的裝置與特定版本的相容性 AWS IoT Greengrass。執行下列命令，將 AWS IoT Greengrass Core 軟體下載至名為 `aws.greengrass.nucleus.zip` 的檔案。將 `##` 取代為 IDT [版本的支援核元件](#) 版本。

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
```

將下載aws.greengrass.nucleus.zip的檔案放在 *<device-tester-extract-location>/products/* 資料夾中。

Note

請勿將相同作業系統和架構的多個檔案放在這個目錄中。

建立和設定 AWS 帳戶

您必須先執行下列步驟，才能使用 AWS IoT Device Tester for AWS IoT Greengrass V2：

1. [設定 AWS 帳戶](#)。如果您已有 AWS 帳戶，請跳到步驟 2。
2. [設定 IDT 的許可](#)。

這些帳戶許可允許 IDT 代表您存取 AWS 服務和建立 AWS 資源，例如 AWS IoT 物件和 AWS IoT Greengrass 元件。

為了建立這些資源，IDT for AWS IoT Greengrass V2 會使用config.json檔案中設定的 AWS 登入資料，代表您進行 API 呼叫。系統會在測試期間的不同時間點內佈建這些資源。

Note

雖然大多數測試都符合 [AWS 免費方案](#) 的資格，但您必須在註冊時提供信用卡 AWS 帳戶。如需詳細資訊，請參閱[如果我的帳戶適用於免費方案，為何需要付款方式？](#)。

步驟 1：設定 AWS 帳戶

在此步驟中，建立和設定 AWS 帳戶。如果您已擁有 AWS 帳戶，請跳到 [the section called “步驟 2：設定 IDT 的許可”](#)。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。
2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時，AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

若要建立管理員使用者，請選擇下列其中一個選項。

選擇一種管理管理員的方式	到	根據	您也可以
在 IAM Identity Center (建議)	使用短期憑證存取 AWS。 這與安全性最佳實務一致。有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務 。	請遵循 AWS IAM Identity Center 使用者指南的 入門 中的說明。	在 AWS Command Line Interface 使用者指南中設定 AWS CLI 以使用來設定 AWS IAM Identity Center 程式設計存取。
在 IAM 中 (不建議使用)	使用長期憑證存取 AWS。	請遵循《IAM 使用者指南》中 建立 IAM 使用者以進行緊急存取 的指示。	請依照《IAM 使用者指南》中的 管理 IAM 使用者的存取金鑰 設定以程式設計方式存取。

步驟 2：設定 IDT 的許可

在此步驟中，設定 IDT for AWS IoT Greengrass V2 用來執行測試和收集 IDT 用量資料的許可。您可以使用 [AWS 管理主控台](#) 或 [AWS Command Line Interface \(AWS CLI\)](#) 來建立 IAM 政策和 IDT 的測試使用者，然後將政策連接到使用者。如果您已為 IDT 建立測試使用者，請跳至 [設定您的裝置以執行 IDT 測試](#)。

設定 IDT (主控台) 的許可

1. 登入 [IAM 主控台](#)。
2. 建立客戶受管政策，該政策授與建立具有特定許可之角色的許可。

- a. 在導覽窗格中，選擇政策，然後選擇建立政策。
- b. 如果您不是使用 PreInstalled，請在 JSON 索引標籤上，將預留位置內容取代為下列政策。如果您使用的是 PreInstalled，請繼續下列步驟。

```
{
  "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. 如果您使用的是 PreInstalled，請在 JSON 標籤上，將預留位置內容取代為下列政策。請確定您：
- 將 `iotResources` 陳述式中的 *thingName* 和 *thingGroup* 取代為測試裝置 (DUT) 上 Greengrass 安裝期間建立的物件名稱和物件群組，以新增許可。
 - 將 `roleAliasResources` 陳述式和 `passRoleForResources` 陳述式中的 *passRole* 和 *roleAlias* 取代為 DUT 上 Greengrass 安裝期間建立的角色。

```

{
  "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/idx-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idx-*",
        "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:*:*:idx-*"
},
{
    "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/idx-*"
    ]
},
{
    "Sid": "idxExecuteAndCollectMetrics",
    "Effect": "Allow",
    "Action": [
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdx",
        "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

如果您想要使用 [自訂 IAM 角色做為待測裝置的字符交換角色](#)，請務必更新政策中的 `roleAliasResources` 陳述式和 `passRoleForResources` 陳述式，以允許自訂 IAM 角色資源。

- d. 選擇檢閱政策。
 - e. 對於名稱，輸入 **IDTGreengrassIAMPermissions**。在 Summary (摘要) 下，檢閱您的政策所授與的許可。
 - f. 選擇建立政策。
3. 建立 IAM 使用者並連接 IDT for 所需的許可 AWS IoT Greengrass。
 - a. 建立 IAM 使用者。請遵循《IAM 使用者指南》中 [建立 IAM 使用者（主控台）](#) 的步驟 1 到 5。
 - b. 將許可連接至您的 IAM 使用者：
 - i. 在 Set permissions (設定許可) 頁面上，選擇 Attach existing policies directly (直接連接現有的政策)。
 - ii. 搜尋您在上一個步驟中建立的 IDTGreengrassIAMPermissions 政策。選取核取方塊。
 - c. 選擇下一步：標籤。
 - d. 選擇 Next: Review (下一步：檢閱) 以檢視選擇的摘要。
 - e. 選擇 Create user (建立使用者)。

- f. 若要檢視使用者的存取金鑰 (存取金鑰 ID 和私密存取金鑰)，請選擇密碼和存取金鑰旁的 Show (顯示)。若要儲存存取金鑰，請選擇 Download.csv，並將檔案儲存到安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。
4. 下一步：設定您的[實體裝置](#)。

步驟 2：設定 IDT (AWS CLI) 的許可

1. 如果尚未安裝，請在電腦上安裝和設定 AWS CLI。請遵循AWS Command Line Interface 《使用者指南》中的[安裝 AWS CLI](#) 中的步驟。

Note

AWS CLI 是開放原始碼工具，可用來與 AWS 命令列 Shell 中的 服務互動。

2. 建立客戶受管政策，以授予管理 IDT 和 AWS IoT Greengrass 角色的許可。
 - a. 如果您不是使用 PreInstalled，請開啟文字編輯器，並將下列政策內容儲存在 JSON 檔案中。如果您使用的是 PreInstalled，請繼續下列步驟。

```
{
  "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. 如果您使用的是 PreInstalled，請開啟文字編輯器，並將下列政策內容儲存在 JSON 檔案中。請確定您：
- 在測試裝置 (DUT) 的 Greengrass 安裝期間建立的 `iotResources` 陳述式中取代 *thingName* 和 *thingGroup*，以新增許可。
 - 將 `roleAliasResources` 陳述式和 `passRoleForResources` 陳述式中的 *passRole* 和 *roleAlias* 取代為 DUT 上 Greengrass 安裝期間建立的角色。

```
{
  "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/idx-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idx-*",
        "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

如果您想要使用 [自訂 IAM 角色做為待測裝置的字符交換角色](#)，請務必更新政策中的 `roleAliasResources` 陳述式和 `passRoleForResources` 陳述式，以允許自訂 IAM 角色資源。

- c. 執行下列命令來建立名為 `的客戶受管政策IDTGreengrassIAMPermissions`。`policy.json` 以您在上一個步驟中建立的 JSON 檔案的完整路徑取代。


```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-document file://policy.json
```

3. 建立 IAM 使用者並連接 IDT for 所需的許可 AWS IoT Greengrass。

- a. 建立 IAM 使用者。在此範例設定中，使用者命名為 IDTGreengrassUser。

```
aws iam create-user --user-name IDTGreengrassUser
```

- b. 將您在步驟 2 中建立 IDTGreengrassIAMPermissions 的政策連接到您的 IAM 使用者。使用的 ID 取代命令中的 *<account-id>* AWS 帳戶。

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. 為使用者建立私密存取金鑰。

```
aws iam create-access-key --user-name IDTGreengrassUser
```

將輸出儲存在安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

5. 下一步：設定您的[實體裝置](#)。

AWS IoT Device Tester 許可

下列政策說明 AWS IoT Device Tester 許可。

AWS IoT Device Tester 需要這些許可才能進行版本檢查和自動更新功能。

- `iot-device-tester:SupportedVersion`

AWS IoT Device Tester 准許擷取支援的產品、測試套件和 IDT 版本清單。

- `iot-device-tester:LatestIdt`

AWS IoT Device Tester 准許擷取可供下載的最新 IDT 版本。

- `iot-device-tester:CheckVersion`

AWS IoT Device Tester 准許檢查 IDT、測試套件和產品的版本相容性。

- `iot-device-tester:DownloadTestSuite`

AWS IoT Device Tester 准許下載測試套件更新。

AWS IoT Device Tester 也會針對選用指標報告使用下列許可：

- `iot-device-tester:SendMetrics`

准許 AWS 收集有關 AWS IoT Device Tester 內部使用的指標。如果省略此許可，則不會收集這些指標。

設定您的裝置以執行 IDT 測試

若要讓 IDT 執行裝置資格測試，您必須設定主機電腦以存取您的裝置，並在裝置上設定使用者許可。

在主機電腦上安裝 Java

從 IDT v4.2.0 開始，的選用資格測試 AWS IoT Greengrass 需要執行 Java。

您可以使用 Java 第 8 版或更新版本。我們建議您使用 [Amazon Corretto](#) 或 [OpenJDK](#) 長期支援版本。第 8 版或更新版本為必要項目。

設定主機電腦以存取待測裝置

IDT 是在您的主機電腦上執行，而且必須能夠使用 SSH 連線到您的裝置。有兩個選項允許 IDT 取得待測裝置的 SSH 存取權：

1. 依照此處的指示來建立 SSH 金鑰對，並授權您的金鑰可以登入待測裝置，無需指定密碼。
2. 提供 `device.json` 檔案中每個裝置的使用者名稱和密碼。如需詳細資訊，請參閱[設定 device.json](#)。

您可以使用任何 SSL 實作來建立 SSH 金鑰。以下指示展示如何使用 [SSH-KEYGEN](#) 或 [PuTTYgen](#) (適用於 Windows)。如果您使用的是另一個 SSL 實作，請參閱該實作的文件。

IDT 使用 SSH 金鑰向待測裝置進行驗證。

使用 SSH-KEYGEN 建立 SSH 金鑰

1. 建立 SSH 金鑰。

您可以使用 Open SSH `ssh-keygen` 命令建立 SSH 金鑰對。如果您的主機電腦上已有 SSH 金鑰對，則最佳實務是特別為 IDT 建立 SSH 金鑰對。如此一來，在您完成測試之後，若沒有輸入密碼，主機電腦再也無法連接至您的裝置。它還可讓您限制只有需要遠端裝置的人，才能存取該裝置。

 Note

Windows 沒有安裝的 SSH 用戶端。如需在 Windows 上安裝 SSH 用戶端的詳細資訊，請參閱[下載 SSH 用戶端軟體](#)。

`ssh-keygen` 命令會提示您提供金鑰對的存放名稱和路徑。根據預設，該金鑰對檔案會命名為 `id_rsa` (私有金鑰) 和 `id_rsa.pub` (公有金鑰)。在 macOS 和 Linux 上，這些檔案的預設位置是 `~/.ssh/`。在 Windows 上，預設位置為 `C:\Users\<user-name>\.ssh`。

出現提示時，請輸入金鑰字詞來保護您的 SSH 金鑰。如需詳細資訊，請參閱[產生新的 SSH 金鑰](#)。

2. 將授權的 SSH 金鑰新增至待測裝置。

IDT 必須使用您的 SSH 私有金鑰登入待測裝置。請從您的主機電腦使用 `ssh-copy-id` 命令，授權您的 SSH 私有金鑰登入待測裝置。此命令會將您的公有金鑰新增至待測裝置上的 `~/.ssh/authorized_keys` 檔案。例如：

```
$ ssh-copy-id <remote-ssh-user>@<remote-device-ip>
```

remote-ssh-user 是用來登入測試裝置的使用者名稱，而 *remote-device-ip* 是要執行測試的測試裝置 IP 地址。例如：

```
ssh-copy-id pi@192.168.1.5
```

出現提示時，請輸入您在 `ssh-copy-id` 命令中指定的使用者名稱密碼。

`ssh-copy-id` 假設公有金鑰名為 `id_rsa.pub`，並存放在預設位置 (macOS 和 Linux 為 `~/.ssh/`，Windows 為 `C:\Users\<user-name>\.ssh`)。如果您給公有金鑰不同的名稱，或將其存放在不同的位置中，則必須在 `ssh-copy-id` 中使用 `-i` 選項，以指定 SSH 公有金鑰的完整路徑 (例如，`ssh-copy-id -i ~/my/path/myKey.pub`)。如需有關建立 SSH 金鑰和複製公有金鑰的詳細資訊，請參閱 [SSH-COPY-ID](#)。

使用 PuTTYgen 建立 SSH 金鑰 (僅限 Windows)

1. 確定您的待測裝置上已安裝 OpenSSH 伺服器 and 用戶端。如需詳細資訊，請參閱 [OpenSSH](#)。
2. 在您的待測裝置上安裝 [PuTTYgen](#)。
3. 開啟 PuTTYgen。
4. 選擇 Generate (產生)，並將滑鼠游標移到方塊內以產生私有金鑰。
5. 從 Conversions (轉換) 功能表中，選擇 Export OpenSSH key (匯出 OpenSSH 金鑰)，然後以 .pem 副檔名儲存私有金鑰。
6. 將公有金鑰新增至待測裝置上的 /home/<user>/.ssh/authorized_keys 檔案。
 - a. 從 PuTTYgen 視窗複製公有金鑰文字。
 - b. 使用 PuTTY 在您的待測裝置上建立工作階段。
 - i. 從命令提示字元或 Windows Powershell 視窗中，執行下列命令：

```
C:/<path-to-putty>/putty.exe -ssh <user>@<dut-ip-address>
```

- ii. 出現提示時，請輸入您裝置的密碼。
 - iii. 使用 vi 或其他文字編輯器，將公有金鑰附加到待測裝置上的 /home/<user>/.ssh/authorized_keys 檔案。
7. 使用您的使用者名稱、IP 地址，以及私有金鑰檔案的路徑 (您剛針對待測裝置將該檔案儲存在主機電腦上) 來更新 device.json 檔案。如需詳細資訊，請參閱 [the section called “設定 device.json”](#)。請務必提供私有金鑰的完整路徑和檔案名稱，並使用正斜線 (/)。例如，若為 Windows 路徑 C:\DT\privatekey.pem，請在 device.json 檔案中使用 C:/DT/privatekey.pem。

設定 Windows 裝置的使用者登入資料

若要符合 Windows 型裝置的資格，您必須在測試裝置上的 LocalSystem 帳戶中為下列使用者設定使用者憑證：

- 預設 Greengrass 使用者 (ggc_user)。
- 您用來連線至待測裝置的使用者。您可以在 [device.json 檔案中](#) 設定此使用者。

您必須在待測裝置的 LocalSystem 帳戶中建立每個使用者，然後將使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。

在 Windows 裝置上設定使用者

1. 以管理員身分開啟 Windows 命令提示字元 (cmd.exe)。
2. 在 Windows 裝置的 LocalSystem 帳戶中建立使用者。針對您要建立的每個使用者執行下列命令。對於預設 Greengrass 使用者，請以取代 `user-name` `ggc_user`。將 `##` 取代為安全密碼。

```
net user /add user-name password
```

3. 從 Microsoft 在裝置上下載並安裝 [PsExec 公用程式](#)。
4. 使用 PsExec 公用程式，將預設使用者的使用者名稱和密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。

針對您要在 Credential Manager 中設定的每個使用者執行下列命令。對於預設 Greengrass 使用者，請以取代 `user-name` `ggc_user`。將 `##` 取代為您先前設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:user-name /user:user-name /pass:password
```

如果 PsExec License Agreement 開啟，請選擇以 Accept 同意授權並執行 命令。

Note

在 Windows 裝置上，LocalSystem 帳戶會執行 Greengrass 核，而且您必須使用 PsExec 公用程式將使用者資訊存放在 LocalSystem 帳戶中。使用 Credential Manager 應用程式會將此資訊存放在目前登入使用者的 Windows 帳戶中，而不是 LocalSystem 帳戶。

在您的裝置上設定使用者許可

IDT 會在待測裝置的各種目錄和檔案上執行操作。其中某些操作需要較高的許可 (使用 sudo)。若要自動化這些操作，IDT for AWS IoT Greengrass V2 必須能夠使用 sudo 執行命令，而不會提示您輸入密碼。

在待測裝置上依照以下步驟，在不提示輸入密碼的情況下允許 sudo 存取。

Note

username 是指 IDT 存取待測裝置時所使用的 SSH 使用者。

將使用者新增至 sudo 群組

1. 在待測裝置上，執行 `sudo usermod -aG sudo <username>`。
2. 登出後再重新登入，以使變更生效。
3. 若要驗證是否已成功新增您的使用者名稱，請執行 `sudo echo test`。如果未提示您輸入密碼，表示已正確設定您的使用者。
4. 開啟 `/etc/sudoers` 檔案，然後在檔案結尾處新增以下一行：

```
<ssh-username> ALL=(ALL) NOPASSWD: ALL
```

設定自訂權杖交換角色

您可以選擇使用自訂 IAM 角色做為待測裝置與 AWS 資源互動時所擔任的權杖交換角色。如需建立 IAM 角色的資訊，請參閱 [《IAM 使用者指南》中的建立 IAM 角色](#)。

您必須符合下列要求，才能允許 IDT 使用您的自訂 IAM 角色。我們強烈建議您只將最低必要政策動作新增至此角色。

- [userdata.json](#) 組態檔案必須更新，才能將 `GreengrassV2TokenExchangeRole` 參數設定為 `true`。
- 自訂 IAM 角色必須使用下列最低信任政策設定：

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "credentials.iot.amazonaws.com",
          "lambda.amazonaws.com",
          "sagemaker.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- 自訂 IAM 角色必須使用下列最低許可政策設定：

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": "*"
    }
  ]
}
```

- 自訂 IAM 角色的名稱必須符合您在測試使用者的 IAM 許可中指定的 IAM 角色資源。根據預設，[測試使用者政策](#)允許存取在其角色名稱中具有 idt-字首的 IAM 角色。如果您的 IAM 角色名稱不使用此字首，請將 `arn:aws:iam::*:role/custom-iam-role-name` 資源新增至 `roleAliasResources` 陳述式和測試使用者政策中的 `passRoleForResources` 陳述式，如下列範例所示：

Example **passRoleForResources** 陳述式

```
{
```

```

    "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** 陳述式

```

{
  "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"
  ]
}

```

設定您的裝置以測試選用功能

本節說明針對選用的 Docker 和機器學習 (ML) 功能執行 IDT 測試的裝置需求。ML 功能僅在 IDT v4.9.3 中受支援。只有當您想要測試這些功能時，才必須確定您的裝置符合這些要求。否則，請繼續進行[the section called “設定 IDT 設定”](#)。

主題

- [Docker 資格要求](#)

- [ML 資格要求](#)
- [HSM 資格要求](#)

Docker 資格要求

IDT for AWS IoT Greengrass V2 提供 Docker 資格測試，以驗證您的裝置是否可以使用 AWS 提供的 [Docker 應用程式管理員](#) 元件來下載您可以使用自訂 Docker 容器元件執行的 Docker 容器映像。如需建立自訂 Docker 元件的詳細資訊，請參閱 [執行 Docker 容器](#)。

若要執行 Docker 資格測試，待測裝置必須符合下列需求，才能部署 Docker 應用程式管理員元件。

- [Docker 引擎](#) 1.9.1 或更新版本安裝在 Greengrass 核心裝置上。20.10 版是經過驗證可與 AWS IoT Greengrass Core 軟體搭配使用的最新版本。您必須先直接在核心裝置上安裝 Docker，才能部署執行 Docker 容器的元件。
- 在部署此元件之前，Docker 協助程式會在核心裝置上啟動和執行。
- 執行 Docker 容器元件的系統使用者必須具有根或管理員許可，或者您必須將 Docker 設定為以非根或非註冊使用者身分執行。
 - 在 Linux 裝置上，您可以將使用者新增至 docker 群組，在沒有 `sudo` 的情況下呼叫 docker 命令。
 - 在 Windows 裝置上，您可以將使用者新增至 docker-users 群組，在沒有管理員權限的情況下呼叫 docker 命令。

Linux or Unix

若要將 `ggc_user` 或您用來執行 Docker 容器元件的非根使用者新增至 docker 群組，請執行下列命令。

```
sudo usermod -aG docker ggc_user
```

如需詳細資訊，請參閱 [以非根使用者的身分管理 Docker](#)。

Windows Command Prompt (CMD)

若要將 `ggc_user` 或您用來執行 Docker 容器元件的使用者新增至 docker-users 群組，請以管理員身分執行下列命令。

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

若要將 `ggc_user` 或您用來執行 Docker 容器元件的使用者新增至 `docker-users` 群組，請以管理員身分執行下列命令。

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

ML 資格要求

Note

機器學習功能僅在 IDT v4.9.3 中受支援。

IDT for AWS IoT Greengrass V2 提供 ML 資格測試，以驗證您的裝置是否可以使用 AWS 提供的 [機器學習元件](#)，使用 [深度學習執行期](#) 或 [TensorFlow Lite](#) ML 架構在本機執行 ML 推論。如需在 Greengrass 裝置上執行 ML 推論的詳細資訊，請參閱 [執行機器學習推論](#)。

若要執行 ML 資格測試，受測裝置必須符合下列要求，才能部署機器學習元件。

- 在執行 Amazon Linux 2 或 Ubuntu 18.04 的 Greengrass 核心裝置上，[GNU C Library](#) (glibc) 2.27 版或更新版本安裝在裝置上。
- 在 Armv7l 裝置上，例如 Raspberry Pi，安裝在裝置上的 OpenCV-Python 相依性。執行下列命令來安裝相依性。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev  
libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 執行 Raspberry Pi OS Bullseye 的 Raspberry Pi 裝置必須符合下列要求：
 - 裝置上已安裝 NumPy 1.22.4 或更新版本。Raspberry Pi OS Bullseye 包含舊版的 NumPy，因此您可以執行下列命令以在裝置上升級 NumPy。

```
pip3 install --upgrade numpy
```

- 在裝置上啟用的舊版攝影機堆疊。Raspberry Pi OS Bullseye 包含預設啟用且不相容的新攝影機堆疊，因此您必須啟用舊版攝影機堆疊。

啟用舊版攝影機堆疊

1. 執行下列命令以開啟 Raspberry Pi 組態工具。

```
sudo raspi-config
```

2. 選取界面選項。
3. 選取舊版攝影機以啟用舊版攝影機堆疊。
4. 重新啟動 Raspberry Pi。

HSM 資格要求

AWS IoT Greengrass 提供 [PKCS#11 提供者元件](#)，可與裝置上的 PKCS 硬體安全模組 (HSM) 整合。HSM 設定取決於您的裝置和您選擇的 HSM 模組。只要提供 [IDT 組態設定](#) 中記錄的預期 HSM 組態，IDT 就會擁有執行此選用功能資格測試所需的資訊。

設定 IDT AWS IoT Greengrass 設定以執行資格套件

執行測試之前，您必須在主機電腦上設定 AWS 登入資料和裝置的設定。

在 config.json 中設定 AWS 登入資料

您必須在 `<device_tester_extract_location>/configs/config.json` 檔案中設定 IAM 使用者登入資料。使用在 中建立的 IDT for AWS IoT Greengrass V2 使用者的登入資料 [the section called “建立和設定 AWS 帳戶”](#)。您可以使用下列兩種方式的其中之一指定登入資料：

- 在登入資料檔案中
- 做為環境變數

使用 AWS 登入資料檔案設定登入資料

IDT 會使用與 AWS CLI 相同的登入資料檔案。如需詳細資訊，請參閱 [組態與登入資料檔案](#)。

登入資料檔案的位置會有所不同，取決於您使用的作業系統：

- macOS, Linux: `~/.aws/credentials`
- Windows : `C:\Users\UserName\.aws\credentials`

以下列格式將您的 AWS 登入資料新增至 `credentials` 檔案：

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

若要將 IDT for AWS IoT Greengrass V2 設定為使用 `credentials` 檔案中的 AWS 登入資料，請編輯 `config.json` 檔案，如下所示：

```
{
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

Note

如果您不使用 default AWS 設定檔，請務必變更 `config.json` 檔案中的設定檔名稱。如需詳細資訊，請參閱 [具名描述檔](#)。

使用環境變數設定 AWS 登入資料

環境變數是由作業系統維護且由系統命令使用的變數。如果您關閉 SSH 工作階段，則不會儲存它們。IDT for AWS IoT Greengrass V2 可以使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 環境變數來存放您的 AWS 登入資料。

若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 `export`：

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要在 Windows 上設定這些變數，請使用 `set`：

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
```

```
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要設定 IDT 來使用環境變數，請在 config.json 檔案中編輯 auth 區段。請見此處範例：

```
{
  "awsRegion": "region",
  "auth": {
    "method": "environment"
  }
}
```

設定 device.json

Note

IDT v4.9.3 支援測試 ml、docker 和 streamManagement 功能。IDT v4.9.4 和更新版本支援測試。docker 如果您不想測試這些功能，請將對應的值設定為 no。

除了 AWS 登入資料之外，IDT for AWS IoT Greengrass V2 還需要執行測試之裝置的相關資訊。範例資訊包括 IP 地址、登入資訊、作業系統和 CPU 架構。

您必須使用位於 `<device_tester_extract_location>/configs/device.json` 中的 device.json 範本提供此資訊：

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

如果 method 是設定為 pki，則指定 privKeyPath
 如果 method 是設定為 password，則指定 password

所有包含值的屬性都是必要的，如下所述：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

sku

可唯一識別測試裝置的英數字元值。SKU 用來追蹤合格的主機板。

Note

如果您想要在 Device AWS Partner Catalog 中列出您的裝置，您在此處指定的 SKU 必須符合您在列出程序中使用的 SKU。

features

包含裝置支援功能的陣列。所有功能都是必要的。

arch

測試執行驗證的支援作業系統架構。有效的值如下：

- x86_64
- armv6l
- armv7l
- aarch64

ml

驗證裝置是否符合使用 AWS 提供的機器學習 (ML) 元件所需的所有必要技術相依性。

啟用此功能也會驗證裝置是否可以使用[深度學習執行期](#)和 [TensorFlow Lite](#) ML 架構來執行 ML 推論。

有效值是 dlr 和 tensorflowlite、或 的任意組合。

docker

驗證裝置是否符合所有必要的技術相依性，以使用 AWS 提供的 Docker 應用程式管理員 (aws.greengrass.DockerApplicationManager) 元件。

啟用此功能也會驗證裝置是否可以從 Amazon ECR 下載 Docker 容器映像。

有效值是 yes 或 的任意組合 no。

streamManagement

驗證裝置是否可以下載、安裝和執行 [AWS IoT Greengrass 串流管理員](#)。

有效值是 yes 或 的任意組合 no。

hsi

驗證裝置是否可以使用存放在硬體安全模組 (HSM) 中的私有金鑰和憑證來驗證 AWS IoT 和 AWS IoT Greengrass 服務的連線。此測試也會驗證 AWS 提供的 [PKCS#11 供應商元件](#) 是否可以使用廠商提供的 PKCS#11 程式庫與 HSM 連接。如需詳細資訊，請參閱 [硬體安全整合](#)。

有效值為 hsm 或 no。

 Note

測試 hsi 僅適用於 IDT v4.9.3 和更新版本。

devices.id

使用者定義的唯一識別符，用於識別要測試的裝置。

devices.operatingSystem

裝置作業系統。支援的值為 Linux 和 Windows。

connectivity.protocol

用來與此裝置通訊的通訊協定。目前，唯一支援的值是 ssh 用於實體裝置。

connectivity.ip

要測試之裝置的 IP 位址。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.port

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.publicKeyPath`

選用。用來驗證待測裝置連線之公有金鑰的完整路徑。

當您指定 `publicKeyPath`，IDT 會在建立與待測裝置的 SSH 連線時驗證裝置的公有金鑰。如果未指定此值，IDT 會建立 SSH 連線，但不會驗證裝置的公有金鑰。

我們強烈建議您指定公有金鑰的路徑，並使用安全方法來擷取此公有金鑰。對於標準命令列型 SSH 用戶端，`known_hosts` 檔案中會提供公有金鑰。如果您指定單獨的公有金鑰檔案，此檔案必須使用與 `known_hosts` 檔案相同的格式，也就是 *ip-address key-type public-key*。如果有多個項目具有相同的 `ip-address`，則 IDT 所使用的金鑰類型項目必須早於檔案中的其他項目。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

connectivity.auth.credentials.user

登入要測試之裝置的使用者名稱。

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

如果 method 是設定為 pki，則指定 privKeyPath
如果 method 是設定為 password，則指定 password

所有包含值的屬性都是必要的，如下所述：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

sku

可唯一識別測試裝置的英數字元值。SKU 用來追蹤合格的主機板。

Note

如果您想要在 Device AWS Partner Catalog 中列出您的裝置，您在此處指定的 SKU 必須符合您在列出程序中使用的 SKU。

features

包含裝置支援功能的陣列。所有功能都是必要的。

arch

測試執行驗證的支援作業系統架構。有效的值如下：

- x86_64
- armv6l
- armv7l
- aarch64

`docker`

驗證裝置是否符合所有必要的技術相依性，以使用 AWS 提供的 Docker 應用程式管理員 (`aws.greengrass.DockerApplicationManager`) 元件。

啟用此功能也會驗證裝置是否可以從 Amazon ECR 下載 Docker 容器映像。

有效值是 `yes` 或 的任意組合 `no`。

`hsi`

驗證裝置是否可以使用存放在硬體安全模組 (HSM) 中的私有金鑰和憑證來驗證 AWS IoT 和 AWS IoT Greengrass 服務的連線。此測試也會驗證 AWS 提供的 [PKCS#11 供應商元件](#) 是否可以使用廠商提供的 PKCS#11 程式庫與 HSM 連接。如需詳細資訊，請參閱[硬體安全整合](#)。

有效值為 `hsm` 或 `no`。

Note

測試 `hsi` 僅適用於 IDT v4.9.3 和更新版本。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`devices.operatingSystem`

裝置作業系統。支援的值為 `Linux` 和 `Windows`。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。目前，唯一支援的值是 `ssh` 用於實體裝置。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.port`

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.publicKeyPath`

選用。用來驗證待測裝置連線之公有金鑰的完整路徑。

當您指定 `publicKeyPath`，IDT 會在建立與待測裝置的 SSH 連線時驗證裝置的公有金鑰。如果未指定此值，IDT 會建立 SSH 連線，但不會驗證裝置的公有金鑰。

我們強烈建議您指定公有金鑰的路徑，並使用安全方法來擷取此公有金鑰。對於標準命令列型 SSH 用戶端，`known_hosts` 檔案中會提供公有金鑰。如果您指定單獨的公有金鑰檔案，此檔案必須使用與 `known_hosts` 檔案相同的格式，也就是 *ip-address key-type public-key*。如果有多個項目具有相同的 `ip-address`，則 IDT 所使用的金鑰類型項目必須早於檔案中的其他項目。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

設定 userdata.json

IDT for AWS IoT Greengrass V2 也需要有關測試成品和 AWS IoT Greengrass 軟體位置的其他資訊。

您必須使用位於 `<device_tester_extract_location>/configs/userdata.json` 中的 `userdata.json` 範本提供此資訊：

```
{
  "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"
  }
}
```

所有包含值的屬性都需要，如下所述：

TempResourcesDirOnDevice

待測裝置上要存放測試成品之暫存資料夾的完整路徑。確定寫入此目錄不需要 `sudo` 許可。

Note

IDT 會在完成執行測試時刪除此資料夾的內容。

InstallationDirRootOnDevice

要安裝之裝置上資料夾的完整路徑 AWS IoT Greengrass。對於 PreInstalled Greengrass，這是 Greengrass 安裝目錄的路徑。

您必須為此資料夾設定必要的檔案許可。對安裝路徑中的每個資料夾執行下列命令。

```
sudo chmod 755 folder-name
```

GreengrassNucleusZip

主機電腦上 Greengrass 核 ZIP (greengrass-nucleus-latest.zip) 檔案的完整路徑。使用 PreInstalled Greengrass 進行測試時，不需要此欄位。

Note

如需 IDT for 的 Greengrass 核支援版本的相關資訊 AWS IoT Greengrass，請參閱 [最新版 IDT for AWS IoT Greengrass V2](#)。若要下載最新的 Greengrass 軟體，請參閱 [下載 AWS IoT Greengrass 軟體](#)。

PreInstalled

此功能僅適用於 Linux 裝置上的 IDT v4.5.8 和更新版本。

(選用) 當值為#時，IDT 會假設中提到的路徑InstallationDirRootOnDevice是安裝 Greengrass 的目錄。

如需如何在裝置上安裝 Greengrass 的詳細資訊，請參閱 [使用自動資源佈建安裝 AWS IoT Greengrass 核心軟體](#)。如果[使用手動佈建安裝](#)，請在手動建立 AWS IoT 物件時包含「將[AWS IoT 物件](#)新增至新的或現有的物件群組」步驟。IDT 假設物件和物件群組是在安裝設定期間建立的。請確定這些值反映在 effectiveConfig.yaml 檔案中。IDT 會在 effectiveConfig.yaml 下檢查檔案<InstallationDirRootOnDevice>/config/effectiveConfig.yaml。

對於使用 HSM 執行測試，請確定 aws.greengrass.crypto.Pkcs11Provider 欄位已在 中更新 effectiveConfig.yaml。

GreengrassV2TokenExchangeRole

(選用) 您想要用作待測裝置假設與 AWS 資源互動之字符交換角色的自訂 IAM 角色。

Note

IDT 會使用此自訂 IAM 角色，而不是在測試執行期間建立預設權杖交換角色。如果您使用自訂角色，您可以更新[測試使用者的 IAM 許可](#)，以排除允許使用者建立和刪除 IAM 角色和政策的 iamResourcesUpdate 陳述式。

如需建立自訂 IAM 角色做為權杖交換角色的詳細資訊，請參閱 [設定自訂權杖交換角色](#)。

hsm

此功能適用於 IDT v4.5.1 和更新版本。

(選用) 使用硬體安全模組 (HSM) 進行測試的 AWS IoT Greengrass 組態資訊。否則，應省略 hsm 屬性。如需詳細資訊，請參閱[硬體安全整合](#)。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

Warning

如果 IDT 和另一個系統共用硬體安全模組，HSM 組態可能會被視為敏感資料。在這種情況下，您可以將這些組態值儲存在參數存放區 SecureString 參數中 AWS，並設定 IDT 在測試執行期間擷取，以避免以純文字形式保護這些組態值。如需詳細資訊，請參閱[???](#)

hsm.greengrassPkcsPluginJar

您下載至 IDT 主機機器之 [PKCS#11 提供者元件](#) 的完整路徑。AWS IoT Greengrass 提供此元件做為 JAR 檔案，您可以在安裝期間下載此元件以指定為佈建外掛程式。您可以將元件 JAR 檔案的最新版本下載為下列 URL：<https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar>。

hsm.pkcs11ProviderLibrary

由硬體安全模組 (HSM) 供應商提供與 HSM 互動之 PKCS#11 程式庫的完整路徑。

hsm.slotId

用於識別您載入金鑰和憑證之 HSM 插槽的插槽 ID。

hsm.slotLabel

用於識別您載入金鑰和憑證之 HSM 插槽的插槽標籤。

`hsm.slotUserPin`

IDT 用來向 HSM 驗證 AWS IoT Greengrass Core 軟體的使用者 PIN 碼。

 Note

基於安全最佳實務，請勿在生產裝置上使用相同的使用者 PIN 碼。

`hsm.keyLabel`

用於識別硬體模組中之金鑰的標籤。金鑰和憑證都必須使用相同的金鑰標籤。

`hsm.preloadedCertificateArn`

AWS IoT 雲端中上傳裝置憑證的 Amazon Resource Name (ARN)。

您先前必須使用 HSM 中的金鑰產生此憑證，將其匯入 HSM，並將其上傳至 AWS IoT 雲端。如需有關產生和匯入憑證的資訊，請參閱 HSM 的文件。

您必須將憑證上傳到您在 [config.json](#) 中提供的相同帳戶和區域。如需將憑證上傳至的詳細資訊 AWS IoT，請參閱《AWS IoT 開發人員指南》中的[手動註冊用戶端憑證](#)。

`hsm.rootCAPath`

(選用) IDT 主機機器上簽署憑證的根憑證授權機構 (CA) 的完整路徑。如果您建立的 HSM 中的憑證不是由 Amazon 根 CA 簽署，則此為必要項目。

從 AWS 參數存放區擷取組態

AWS IoT Device Tester (IDT) 包含從 [AWS Systems Manager 參數存放區](#) 擷取組態值的選用功能。AWS 參數存放區允許安全且加密的組態儲存。設定後，IDT 可以從 AWS 參數存放區擷取參數，取代將參數以純文字形式儲存在 `userdata.json` 檔案內。這對於應加密儲存的任何敏感資料很有用，例如：密碼、接腳和其他秘密。

1. 若要使用此功能，您必須更新建立 [IDT 使用者](#) 所用的許可，以允許對 IDT 設定為使用的參數執行 `GetParameter` 動作。以下是可新增至 IDT 使用者的許可陳述式範例。如需詳細資訊，請參閱 [AWS Systems Manager 使用者指南](#)。

```
{
```

```

    "Sid": "parameterStoreResources",
    "Effect": "Allow",
    "Action": [
        "ssm:GetParameter"
    ],
    "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
}

```

上述許可設定為允許 IDT 使用萬用字元 擷取名稱開頭為 的所有參數*。您應該根據您的需求自訂此項目，以便 IDT 能夠根據您正在使用的參數命名來擷取任何設定的參數。

2. 您需要將組態值存放在 Parameter Store AWS 中。這可以從 AWS 主控台或 CLI AWS 完成。AWS 參數存放區可讓您選擇加密或未加密的儲存。對於儲存機密、密碼和 PIN 等敏感值，您應該使用加密選項，這是 SecureString 的參數類型。若要使用 CLI AWS 上傳參數，您可以使用下列命令：

```

aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
SecureString

```

您可以使用下列命令來驗證參數是否已存放。（選用）使用 --with-decryption 旗標來擷取解密的 SecureString 參數。

```

aws ssm get-parameter --name IDT-example-name

```

使用 AWS CLI 將上傳目前 CLI 使用者 AWS 區域中的 參數，IDT 將從 中設定的區域擷取參數 config.json。若要從 AWS CLI 檢查您的區域，請使用下列項目：

```

aws configure get region

```

3. 一旦您在 AWS 雲端中有組態值，您就可以更新 IDT 組態內的任何值，以從 AWS 雲端擷取。若要這樣做，請在表單的 IDT 組態中使用預留位置 {{AWS.Parameter.parameter_name}}，以該名稱從參數存放區擷取 AWS 參數。

例如，假設您想要使用步驟 2 中的 IDT-example-name 參數做為 HSM 組態中的 HSM keyLabel。若要這樣做，您可以更新 userdata.json，如下所示：

```

"hsm": {
    "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
    [...]
}

```

IDT 將在步驟 2 IDT-example-value 中設定為 的執行時間擷取此參數的值。此組態類似於設定，"keyLabel": "IDT-example-value" 但該值會以加密方式儲存在 AWS 雲端中。

執行 AWS IoT Greengrass 資格套件

在您 [設定必要組態](#) 之後，便可開始測試。完整測試套件的執行時間取決於您的硬體。做為參考，在 Raspberry Pi 3B 上完成整個測試套件約需 30 分鐘。

使用下列 run-suite 命令來執行一組測試。

```
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
```

所有選項都是選用的。例如，pool-id 如果您只有一個裝置集區，這是一組相同的裝置，在您的 device.json 檔案中定義，您可以省略。或者，如果您想要執行 tests 資料夾中最新的測試套件版本，則可以省略 suite-id。

Note

如果線上有較新的測試套件版本，IDT 會提示您。如需詳細資訊，請參閱 [the section called “測試套件版本”](#)。

執行資格套件的範例命令

下列命令列範例示範如何執行裝置集區的資格測試。如需 run-suite 和其他 IDT 命令的詳細資訊，請參閱 [the section called “IDT 命令”](#)。

使用下列命令，在指定的測試套件中執行所有測試群組。list-suites 命令會列出 tests 資料夾中的測試套件。

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
  --suite-id GGV2Q_1.0.0 \
```

```
--pool-id <pool-id> \  
--userdata userdata.json
```

使用以下命令在測試套件中執行特定測試群組。list-groups 命令會列出測試套件中的測試群組。

```
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
```

使用下列命令在測試群組中執行特定測試案例。

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
--group-id <group-id> \  
--test-id <test-id> \  
--userdata userdata.json
```

使用以下命令在測試群組中執行多個測試案例。

```
devicetester_[linux | mac | win]_x86-64 run-suite \  
--group-id <group-id> \  
--test-id <test-id1>,<test-id2> \  
--userdata userdata.json
```

使用下列命令列出測試群組中的所有測試案例。

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

我們建議您執行完整的資格測試套件，以正確的順序執行測試群組相依性。如果您選擇執行特定測試群組，建議您先執行相依性檢查程式測試群組，以確保在執行相關測試群組之前已安裝所有 Greengrass 相依性。例如：

- 在執行核心資格測試組之前執行 coredependencies。

IDT for AWS IoT Greengrass V2 命令

IDT 命令位於 *<device-tester-extract-location>*/bin 目錄中。若要執行測試套件，請以下列格式提供 命令：

help

列出所指定命令的相關資訊。

list-groups

列出指定測試套件中的群組。

list-suites

列出可用的測試套件。

list-supported-products

在此情況下，列出支援的 產品 AWS IoT Greengrass 版本，以及目前 IDT 版本的測試套件版本。

list-test-cases

列出特定測試群組中的測試案例。支援下列選項：

- group-id。要搜尋的測試群組。此選項為必要選項，且必須指定單一群組。

run-suite

在裝置集區上執行測試套件。以下是一些支援的選項：

- suite-id。要執行的測試套件版本。如果未指定，IDT 會使用 tests 資料夾中的最新版本。
- group-id。要執行的測試群組，以逗號分隔的清單。如果未指定，IDT 會根據 中的設定，在測試套件中執行所有適當的測試群組device.json。IDT 不會根據您設定的設定執行裝置不支援的任何測試群組，即使這些測試群組是在group-id清單中指定。
- test-id。要執行的測試案例，以逗號分隔的清單。指定時，group-id 必須指定單一群組。
- pool-id。要測試的裝置集區。如果 device.json 檔案中已定義多個裝置集區，則必須指定集區。
- stop-on-first-failure。設定 IDT 在第一次失敗時停止執行。當您想要偵錯指定的測試群組group-id時，請使用此選項搭配 。執行完整測試套件產生資格報告時，請勿使用此選項。
- update-idt。設定提示更新 IDT 的回應。如果 IDT 偵測到有較新的版本，Y回應會停止測試執行。N 回應會繼續測試執行。
- userdata。userdata.json 檔案的完整路徑，其中包含測試成品路徑的相關資訊。該run-suite命令需要此選項。userdata.json 檔案必須位於 *devicetester_extract_location*/devicetester_ggv2_#win/mac/linux#/configs/ 目錄中。

如需 run-suite 選項的詳細資訊，請使用下列 help 選項：

```
devicetester_[linux | mac | win]_x86-64 run-suite -h
```

了解結果和日誌

本節說明如何檢視和解譯 IDT 結果報告與日誌。

若要對錯誤進行故障診斷，請參閱 [對 IDT for AWS IoT Greengrass V2 進行故障診斷](#)。

檢視結果

執行期間，IDT 會將錯誤寫入主控台、日誌檔和測試報告。IDT 完成資格測試套件後會產生兩份測試報告。這些報告位於中 `<device-tester-extract-location>/results/<execution-id>/`。兩個報告都會擷取執行資格測試套件的結果。

awsiotdevicetester_report.xml 是您提交給 AWS 以在 Device AWS Partner Catalog 中列出裝置的資格測試報告。該報告包含下列元素：

- IDT 版本。
- 已測試的 AWS IoT Greengrass 版本。
- device.json 檔案中指定的 SKU 和裝置集區名稱。
- device.json 檔案中指定的裝置集區的功能。
- 測試結果的彙總摘要。
- 根據裝置功能進行測試的程式庫測試結果明細，例如本機資源存取、陰影和 MQTT。

GGV2Q_Result.xml 報告採用 [JUnit XML 格式](#)。您可以將它整合到持續整合和部署平台，例如 [Jenkins](#)、[Bamboo](#) 等。該報告包含下列元素：

- 測試結果的彙總摘要。
- 已測試 AWS IoT Greengrass 功能對測試結果進行分解。

解譯 AWS IoT Device Tester 結果

awsiotdevicetester_report.xml 或 awsiotdevicetester_report.xml 的報告區段會列出已執行的測試及結果。

第一個 XML 標籤 <testsuites> 包含測試執行的摘要。例如：

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0" disabled="0">
```

<testsuites> 標籤中使用的屬性

name

測試套件的名稱。

time

執行資格套件所需的時間，以秒為單位。

tests

執行的測試數量。

failures

已執行但未通過的測試次數。

errors

IDT 無法執行的測試數量。

disabled

忽略此屬性。不會使用。

awsiotdevicetester_report.xml 檔案包含 <awsproduct> 標籤，其中包含關於受測產品和經過一系列測試驗證後之產品功能的資訊。

<awsproduct> 標籤中使用的屬性

name

受測產品名稱。

version

受測產品版本。

features

驗證的功能。標記為 `required` 的功能為提交主機板獲得資格時所需。以下片段顯示此資訊如何出現在 `awsiotdevicetester_report.xml` 檔案中。

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></feature>
```

如果必要功能沒有測試失敗或錯誤，您的裝置會符合執行的技術需求，AWS IoT Greengrass 並且可以與 AWS IoT 服務互通。如果您想要在 Device AWS Partner Catalog 中列出您的裝置，您可以使用此報告做為資格證據。

如果測試發生失敗或錯誤，您可以檢閱 `<testsuites>` XML 標籤來識別失敗的測試。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試結果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0" errors="0" skipped="0">
```

其格式類似於 `<testsuites>` 標籤，但有不使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤中，每個為測試群組執行的測試都有 `<testcase>` 標籤。例如：

```
<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>>
```

`<testcase>` 標籤中使用的屬性

name

測試的名稱。

attempts

IDT 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 `<failure>` 或 `<error>` 標籤新增至 `<testcase>` 標籤，其中附有相關資訊以利故障診斷。例如：

```
<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>
```

檢視日誌

IDT 會從 中的測試執行產生日誌 `<devicetester-extract-location>/results/<execution-id>/logs`。該工具會產生兩組日誌：

`test_manager.log`

從 的測試管理員元件產生的日誌 AWS IoT Device Tester（例如，與組態、測試排序和報告產生相關的日誌）。

`<test-case-id>.log`（for example, `lambdaDeploymentTest.log`）

測試群組內的測試案例日誌，包括測試中裝置的日誌。從 IDT v4.2.0 開始，IDT 會將每個測試案例的測試日誌分組到 `<devicetester-extract-location>/results/<execution-id>/logs/<test-group-id>/`目錄中的個別 `<test-case-id>` 資料夾。

使用 IDT 來開發和執行您自己的測試套件

從 IDT v4.0.1 開始，IDT for AWS IoT Greengrass V2 結合了標準化的組態設定和結果格式，以及測試套件環境，可讓您為裝置和裝置軟體開發自訂測試套件。您可以為自己的內部驗證新增自訂測試，或將其提供給客戶進行裝置驗證。

使用 IDT 來開發和執行自訂測試套件，如下所示：

開發自訂測試套件

- 為您要測試的 Greengrass 裝置建立具有自訂測試邏輯的測試套件。
- 提供 IDT 您的自訂測試套件來測試執行器。包含測試套件特定設定組態的相關資訊。

執行自訂測試套件

- 設定您要測試的裝置。
- 實作您想要使用的測試套件所需的設定組態。
- 使用 IDT 執行您的自訂測試套件。
- 檢視 IDT 所執行測試的測試結果和執行日誌。

下載 AWS IoT Device Tester 適用於 的最新版本 AWS IoT Greengrass

下載[最新版本](#)的 IDT，並將軟體解壓縮到檔案系統上具有讀取/寫入許可的位置 (`<device-tester-extract-location>`)。

Note

IDT 不支援由多位使用者從共用位置執行，例如 NFS 目錄或 Windows 網路共用資料夾。我們建議您將 IDT 套件解壓縮到本機磁碟機，並在本機工作站上執行 IDT 二進位檔。

Windows 的路徑長度限制為 260 個字元。如果您使用的是 Windows，請將 IDT 解壓縮到根目錄，例如 C:\ 或 D:\，使路徑保持在 260 個字元的限制以下。

測試套件建立工作流程

測試套件由三種類型的檔案組成：

- 為 IDT 提供如何執行測試套件相關資訊的組態檔案。
- 測試 IDT 用來執行測試案例的可執行檔。
- 執行測試所需的其他檔案。

完成下列基本步驟以建立自訂 IDT 測試：

1. 為您的測試套件[建立組態檔案](#)。
2. [建立包含測試套件測試邏輯的測試案例可執行檔](#)。
3. 驗證並記錄[測試執行器執行測試套件所需的組態資訊](#)。
4. 確認 IDT 可以執行您的測試套件，並如預期產生[測試結果](#)。

若要快速建置範例自訂套件並進行執行，請遵循 中的指示[教學課程：建置並執行範例 IDT 測試套件](#)。

若要開始在 Python 中建立自訂測試套件，請參閱 [教學課程：開發簡單的 IDT 測試套件](#)。

教學課程：建置並執行範例 IDT 測試套件

AWS IoT Device Tester 下載包含範例測試套件的原始程式碼。您可以完成此教學課程來建置和執行範例測試套件，以了解如何使用 IDT for AWS IoT Greengrass 來執行自訂測試套件。

在本教學課程中，您將完成下列步驟：

1. [建置範例測試套件](#)
2. [使用 IDT 執行範例測試套件](#)

先決條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
 - 的最新版本 AWS IoT Device Tester
 - [Python](#) 3.7 或更新版本

若要檢查安裝在電腦上的 Python 版本，請執行下列命令：

```
python3 --version
```

在 Windows 上，如果使用此命令傳回錯誤，請將 `python3` 改用 `python`。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行下列命令，將 `python3` 設定為 `python` 命令的別名。

```
Set-Alias -Name "python3" -Value "python"
```

如果未傳回版本資訊或版本編號小於 3.7，請遵循[下載 Python](#) 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 [Python 文件](#)。

- [urllib3](#)

若要驗證 `urllib3` 是否正確安裝，請執行下列命令：

```
python3 -c 'import urllib3'
```

如果 `urllib3` 未安裝，請執行下列命令來安裝它：

```
python3 -m pip install urllib3
```

- 裝置要求
 - 具有 Linux 作業系統和與主機電腦相同網路連線的裝置。

我們建議您將 [Raspberry Pi](#) 與 Raspberry Pi 作業系統搭配使用。請務必在 Raspberry Pi 上設定 [SSH](#) 以遠端連線到它。

設定 IDT 的裝置資訊

設定您的裝置資訊，讓 IDT 執行測試。您必須使用下列資訊更新位於 *<device-tester-extract-location>*/configs 資料夾的 device.json 範本。

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

在 devices 物件中，提供下列資訊：

id

您裝置的使用者定義唯一識別符。

connectivity.ip

您裝置的 IP 地址。

`connectivity.port`

選用。用於 SSH 連線至您裝置的連接埠號碼。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.user`

用來登入裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用來登入您裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`devices.connectivity.auth.credentials.password`

用於登入您裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

Note

如果 `method` 是設定為 `pki`，則指定 `privKeyPath`

如果 `method` 是設定為 `password`，則指定 `password`

建置範例測試套件

`<device-tester-extract-location>/samples/python` 資料夾包含範例組態檔案、原始程式碼和 IDT 用戶端 SDK，您可以使用提供的建置指令碼將其合併為測試套件。下列目錄樹狀目錄顯示這些範例檔案的位置：

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

若要建置測試套件，請在主機電腦上執行下列命令：

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
```

這會在 IDTSampleSuitePython_1.0.0 資料夾內的 `<device-tester-extract-location>/tests` 資料夾中建立範例測試套件。檢閱 IDTSampleSuitePython_1.0.0 資料夾中的檔案，以了解範例測試套件的結構，並查看各種測試案例可執行檔和測試組態 JSON 檔案的範例。

 Note

範例測試套件包含 python 原始程式碼。請勿在測試套件程式碼中包含敏感資訊。

下一步：使用 IDT 執行[您建立的範例測試套件](#)。

使用 IDT 執行範例測試套件

若要執行範例測試套件，請在主機電腦上執行下列命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT 會執行範例測試套件，並將結果串流至主控台。當測試完成執行時，您會看到下列資訊：

```
===== 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
```

疑難排解

使用以下資訊來協助解決完成教學課程的任何問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤日誌串流至主控台，以協助您對測試執行進行疑難排解。請確定您符合本教學課程的所有[先決條件](#)。

無法連線至待測裝置

請確認下列內容：

- 您的 `device.json` 檔案包含正確的 IP 地址、連接埠和身分驗證資訊。
- 您可以從主機電腦透過 SSH 連線至裝置。

教學課程：開發簡單的 IDT 測試套件

測試套件結合了下列項目：

- 包含測試邏輯的測試可執行檔
- 描述測試套件的組態檔案

本教學課程說明如何使用 IDT for AWS IoT Greengrass 來開發包含單一測試案例的 Python 測試套件。在本教學課程中，您將完成下列步驟：

1. [建立測試套件目錄](#)
2. [建立組態檔案](#)
3. [建立測試案例可執行檔](#)
4. [執行測試套件](#)

先決條件

為了完成本教學，您需要以下項目：

- 主機電腦需求
 - 的最新版本 AWS IoT Device Tester
 - [Python](#) 3.7 或更新版本

若要檢查安裝在電腦上的 Python 版本，請執行下列命令：

```
python3 --version
```


在 Windows 上，如果使用此命令傳回錯誤，請 `python --version` 改用。如果傳回的版本編號為 3.7 或更新版本，請在 Powershell 終端機中執行下列命令，將 `python3` 設定為 `python` 命令的別名。

```
Set-Alias -Name "python3" -Value "python"
```

如果未傳回版本資訊或版本編號小於 3.7，請遵循[下載 Python](#) 中的指示安裝 Python 3.7+。如需詳細資訊，請參閱 [Python 文件](#)。

- [urllib3](#)

若要驗證 `urllib3` 是否正確安裝，請執行下列命令：

```
python3 -c 'import urllib3'
```

如果 `urllib3` 未安裝，請執行下列命令來安裝它：

```
python3 -m pip install urllib3
```

- 裝置要求
 - 具有 Linux 作業系統和與主機電腦相同網路連線的裝置。

我們建議您將 [Raspberry Pi](#) 與 Raspberry Pi 作業系統搭配使用。請確定您在 Raspberry Pi 上設定 [SSH](#) 以遠端連線到它。

建立測試套件目錄

IDT 會在邏輯上將測試案例分成每個測試套件內的測試群組。每個測試案例都必須位於測試群組內。在本教學課程中，請建立名為 `MyTestSuite_1.0.0` 的資料夾，並在此資料夾中建立下列目錄樹狀目錄：

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

建立組態檔案

您的測試套件必須包含下列必要的[組態檔案](#)：

必要的組態檔案

suite.json

包含測試套件的相關資訊。請參閱 [設定 suite.json](#)。

group.json

包含測試群組的相關資訊。您必須為測試套件中的每個測試群組建立group.json檔案。請參閱 [設定 group.json](#)。

test.json

包含測試案例的相關資訊。您必須為測試套件中的每個測試案例建立test.json檔案。請參閱 [設定 test.json](#)。

1. 在 MyTestSuite_1.0.0/suite資料夾中，建立具有下列結構suite.json的檔案：

```
{
  "id": "MyTestSuite_1.0.0",
  "title": "My Test Suite",
  "details": "This is my test suite.",
  "userDataRequired": false
}
```

2. 在 MyTestSuite_1.0.0/myTestGroup資料夾中，建立具有下列結構group.json的檔案：

```
{
  "id": "MyTestGroup",
  "title": "My Test Group",
  "details": "This is my test group.",
  "optional": false
}
```

3. 在 MyTestSuite_1.0.0/myTestGroup/myTestCase資料夾中，建立具有下列結構test.json的檔案：

```
{
  "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"
      ]
    }
  }
}
```

資料夾的目錄樹狀目錄現在MyTestSuite_1.0.0看起來應該如下所示：

```
MyTestSuite_1.0.0
### suite
### suite.json
### myTestGroup
### group.json
### myTestCase
### test.json
```

取得 IDT 用戶端 SDK

您可以使用 [IDT 用戶端 SDK](#) 讓 IDT 與待測裝置互動，並報告測試結果。在本教學課程中，您將使用開發套件的 Python 版本。

從 *<device-tester-extract-location>/sdks/python/* 資料夾，將 `idt_client` 資料夾複製到您的 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 資料夾。

若要確認軟體開發套件已成功複製，請執行下列命令。

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

建立測試案例可執行檔

測試案例可執行檔包含您要執行的測試邏輯。測試套件可以包含多個測試案例可執行檔。在本教學課程中，您只會建立一個測試案例可執行檔。

1. 建立測試套件檔案。

在 `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` 資料夾中，建立具有下列內容 `myTestCase.py` 的檔案：

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

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

2. 使用用戶端 SDK 函數將下列測試邏輯新增至您的 `myTestCase.py` 檔案：

a. 在待測裝置上執行 SSH 命令。

```
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. 將測試結果傳送至 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()
```

設定 IDT 的裝置資訊

設定您的裝置資訊，讓 IDT 執行測試。您必須使用下列資訊更新位於 *<device-tester-extract-location>*/configs 資料夾的 device.json 範本。

```
[
  {
    "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>"
      }
    }
  }
}
```

在 devices 物件中，提供下列資訊：

id

您裝置的使用者定義唯一識別符。

connectivity.ip

您裝置的 IP 地址。

connectivity.port

選用。用於裝置 SSH 連線的連接埠號碼。

connectivity.auth

連線的驗證資訊。

只有當 connectivity.protocol 設為 ssh 時，才會套用此屬性。

connectivity.auth.method

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- pki

- password

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.user`

用來登入裝置的使用者名稱。

`connectivity.auth.credentials.privKeyPath`

用來登入您裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`devices.connectivity.auth.credentials.password`

用於登入您裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

Note

如果 `method` 是設定為 `pki`，則指定 `privKeyPath`

如果 `method` 是設定為 `password`，則指定 `password`

執行測試套件

建立測試套件之後，您想要確保其如預期般運作。完成下列步驟，以使用您現有的裝置集區執行測試套件。

1. 將您的 `MyTestSuite_1.0.0` 資料夾複製到 `<device-tester-extract-location>/tests`。
2. 執行下列命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT 會執行您的測試套件，並將結果串流至主控台。當測試完成執行時，您會看到下列資訊：

```
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
```

疑難排解

使用以下資訊來協助解決完成教學課程的任何問題。

測試案例未成功執行

如果測試未成功執行，IDT 會將錯誤日誌串流至主控台，以協助您對測試執行進行疑難排解。檢查錯誤日誌之前，請確認下列事項：

- IDT 用戶端 SDK 位於正確的資料夾中，如[此步驟](#)所述。
- 您符合本教學課程的所有[先決條件](#)。

無法連線至待測裝置

請確認下列內容：

- 您的 `device.json` 檔案包含正確的 IP 地址、連接埠和身分驗證資訊。
- 您可以從主機電腦透過 SSH 連線至裝置。

建立 IDT 測試套件組態檔案

本節說明當您撰寫自訂測試套件時，建立組態檔案的格式。

必要的組態檔案

`suite.json`

包含測試套件的相關資訊。請參閱 [設定 `suite.json`](#)。

`group.json`

包含測試群組的相關資訊。您必須為測試套件中的每個測試群組建立 `group.json` 檔案。請參閱 [設定 `group.json`](#)。

`test.json`

包含測試案例的相關資訊。您必須為測試套件中的每個測試案例建立 `test.json` 檔案。請參閱 [設定 `test.json`](#)。

選用組態檔案

`test_orchestrator.yaml` 或 `state_machine.json`

定義 IDT 執行測試套件時如何執行測試。SSE [設定 `test\_orchestrator.yaml`](#)。

Note

從 IDT v4.5.1 開始，您可以使用 `test_orchestrator.yaml` 檔案來定義測試工作流程。在舊版的 IDT 中，您會使用 `state_machine.json` 檔案。如需狀態機器的相關資訊，請參閱 [設定 IDT 狀態機器](#)。

`userdata_schema.json`

定義測試執行器可以包含在其設定組態中的 [`userdata.json` 檔案](#) 結構描述。

此 `userdata.json` 檔案用於執行測試所需的任何其他組態資訊，但不會出現在 `device.json` 檔案中。請參閱 [設定 `userdata\_schema.json`](#)。

組態檔案會放在您的 中 `<custom-test-suite-folder>`，如下所示。

```
<custom-test-suite-folder>
### suite
### suite.json
### test_orchestrator.yaml
### userdata_schema.json
### <test-group-folder>
### group.json
### <test-case-folder>
### test.json
```

設定 suite.json

suite.json 檔案會設定環境變數，並判斷是否需要使用者資料才能執行測試套件。使用下列範本來設定您的 `<custom-test-suite-folder>/suite/suite.json` 檔案：

```
{
  "id": "<suite-name>_<suite-version>",
  "title": "<suite-title>",
  "details": "<suite-details>",
  "userDataRequired": true | false,
  "environmentVariables": [
    {
      "key": "<name>",
      "value": "<value>",
    },
    ...
    {
      "key": "<name>",
      "value": "<value>",
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試套件的唯一使用者定義 ID。的值id必須符合 suite.json 檔案所在的測試套件資料夾名稱。套件名稱和套件版本也必須符合下列要求：

- `<suite-name>` 不能包含底線。

- `<suite-version>` 表示為 `x.x.x`，其中 x 是數字。

ID 會顯示在 IDT 產生的測試報告中。

title

此測試套件所測試之產品或功能的使用者定義名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試套件用途的簡短描述。

userDataRequired

定義測試執行器是否需要在 `userdata.json` 檔案中包含自訂資訊。如果您將此值設定為 `true`，您還必須在測試套件資料夾中包含 [userdata_schema.json 檔案](#)。

environmentVariables

選用。要為此測試套件設定的環境變數陣列。

environmentVariables.key

環境變數的名稱。

environmentVariables.value

環境變數的值。

設定 group.json

`group.json` 檔案會定義測試群組為必要或選用。使用下列範本來設定您的 `<custom-test-suite-folder>/suite/<test-group>/group.json` 檔案：

```
{
  "id": "<group-id>",
  "title": "<group-title>",
  "details": "<group-details>",
  "optional": true | false,
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試群組的唯一使用者定義 ID。的值id必須符合group.json檔案所在的測試群組資料夾名稱，且不能包含底線(_)。ID 用於 IDT 產生的測試報告。

title

測試群組的描述性名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試群組用途的簡短描述。

optional

選用。設定為 true，在 IDT 完成執行必要的測試後，將此測試群組顯示為選用群組。預設值為 false。

設定 test.json

test.json 檔案會決定測試案例可執行檔和測試案例所使用的環境變數。如需建立測試案例可執行檔的詳細資訊，請參閱 [建立 IDT 測試案例可執行檔](#)。

使用下列範本來設定您的<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json檔案：

```
{
  "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>",
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

測試案例的唯一使用者定義 ID。的值id必須符合test.json檔案所在的測試案例資料夾名稱，且不能包含底線(_)。ID 用於 IDT 產生的測試報告。。

title

測試案例的描述性名稱。名稱會顯示在測試執行器的 IDT CLI 中。

details

測試案例用途的簡短描述。

requireDUT

選用。true 如果需要裝置執行此測試，請將 設定為 true，否則將 設定為 false。預設值為 true。測試執行器會設定其用來在device.json檔案中執行測試的裝置。

requiredResources

選用。陣列，提供執行此測試所需的資源裝置相關資訊。

requiredResources.name

執行此測試時為資源裝置提供的唯一名稱。

requiredResources.features

使用者定義資源裝置功能的陣列。

requiredResources.features.name

功能的名稱。您想要使用此裝置的裝置功能。此名稱會與 resource.json 檔案中測試執行器所提供的功能名稱相符。

requiredResources.features.version

選用。功能的版本。此值會與 resource.json 檔案中測試執行器所提供的功能版本相符。如果未提供版本，則不會檢查此功能。如果功能不需要版本號碼，請將此欄位保留空白。

requiredResources.features.jobSlots

選用。此功能可支援的同時測試數量。預設值為 1。如果您希望 IDT 對個別功能使用不同的裝置，建議您將此值設定為 1。

execution.timeout

IDT 等待測試完成執行的時間量（以毫秒為單位）。如需設定此值的詳細資訊，請參閱 [建立 IDT 測試案例可執行檔](#)。

execution.os

根據執行 IDT 之主機電腦的作業系統執行的測試案例可執行檔。支援的值為 linux、mac 和 win。

execution.os.cmd

您想要在指定作業系統上執行的測試案例可執行檔路徑。此位置必須位於系統路徑中。

`execution.os.args`

選用。要用來執行測試案例可執行檔的引數。

`environmentVariables`

選用。此測試案例設定的環境變數陣列。

`environmentVariables.key`

環境變數的名稱。

`environmentVariables.value`

環境變數的值。

Note

如果您在 `test.json` 檔案和 `suite.json` 檔案中指定相同的環境變數，則 `test.json` 檔案的值優先。

設定 `test_orchestrator.yaml`

測試協調器是一種控制測試套件執行流程的建構。它決定測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，並繼續轉換這些狀態，直到達到結束狀態為止。

如果您的測試套件不包含使用者定義的測試協調程式，IDT 會為您產生測試協調程式。

預設測試協調器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 如果未選取特定測試群組，會以隨機順序在測試套件中執行每個測試群組。
- 產生報告並列印主控台摘要，以顯示每個測試群組和測試案例的測試結果。

如需有關 IDT 測試協調器如何運作的詳細資訊，請參閱 [設定 IDT 測試協調器](#)。

設定 `userdata_schema.json`

`userdata_schema.json` 檔案會決定測試執行器提供使用者資料的結構描述。如果您的測試套件需要 `device.json` 檔案中不存在的資訊，則需要使用者資料。例如，您的測試可能需要 Wi-Fi 網路登入資料、特定開放連接埠或使用者必須提供的憑證。此資訊可以做為名為 `inputs` 的輸入參數

提供給 IDT，這是使用者在其 `<device-tester-extract-location>/config` 資料夾中建立 `userdata.json` 的檔案 `userdata` 值。檔案的格式 `userdata.json` 是根據您在測試套件中包含的 `userdata_schema.json` 檔案。

若要指出測試執行器必須提供 `userdata.json` 檔案：

1. 在 `suite.json` 檔案中，將 `userDataRequired` 設定為 `true`。
2. 在您的 `<custom-test-suite-folder>` 中，建立 `userdata_schema.json` 檔案。
3. 編輯 `userdata_schema.json` 檔案以建立有效的 [IETF 草稿 v4 JSON 結構描述](#)。

當 IDT 執行您的測試套件時，其會自動讀取結構描述，並使用它來驗證測試執行器所提供的 `userdata.json` 檔案。如果有效，`userdata.json` 檔案的內容可在 [IDT 內容](#) 和 [測試協調器內容](#) 中使用。

設定 IDT 測試協調器

從 IDT v4.5.1 開始，IDT 包含新的測試協調器元件。測試協調器是一種 IDT 元件，可控制測試套件執行流程，並在 IDT 完成執行所有測試後產生測試報告。測試協調器會根據使用者定義的規則，決定測試選擇和執行測試的順序。

如果您的測試套件不包含使用者定義的測試協調程式，IDT 會為您產生測試協調程式。

預設測試協調器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 如果未選取特定測試群組，會以隨機順序在測試套件中執行每個測試群組。
- 產生報告並列印主控台摘要，以顯示每個測試群組和測試案例的測試結果。

測試協調器會取代 IDT 測試協調器。我們強烈建議您使用測試協調器來開發測試套件，而不是 IDT 測試協調器。測試協調器提供下列改進功能：

- 相較於 IDT 狀態機器使用的強制性格式，會使用宣告格式。這可讓您指定要執行哪些測試，以及何時要執行這些測試。
- 管理特定群組處理、報告產生、錯誤處理和結果追蹤，因此您不需要手動管理這些動作。
- 使用 YAML 格式，預設支援註解。
- 需要比測試協調器少 80% 的磁碟空間，才能定義相同的工作流程。
- 新增測試前驗證，以確認您的工作流程定義不包含不正確的測試 IDs 或循環相依性。

測試協調器格式

您可以使用下列範本來設定自己的 `<custom-test-suite-folder>/suite/test_orchestrator.yaml` 檔案：

```
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
```

如下所述，包含值的所有欄位皆為必要：

Aliases

選用。映射到內容表達式的使用者定義字串。別名可讓您產生易記的名稱，以識別測試協調器組態中的內容表達式。如果您要建立複雜內容表達式或您在多個位置使用的表達式，這特別有用。

您可以使用內容表達式來存放內容查詢，讓您從其他 IDT 組態存取資料。如需詳細資訊，請參閱[存取內容中的資料](#)。

Example 範例

```
Aliases:
  FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
  BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
```

```
FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

ConditionalTests

選用。條件清單，以及在滿足每個條件時執行的對應測試案例。每個條件可以有多個測試案例；不過，您可以將指定的測試案例指派給一個條件。

根據預設，IDT 會執行未指派給此清單中條件的任何測試案例。如果您未指定本節，IDT 會在測試套件中執行所有測試群組。

ConditionalTests 清單中的每個項目都包含下列參數：

Condition

評估為布林值的內容表達式。如果評估的值為 true，IDT 會執行 Tests 參數中指定的測試案例。

Tests

測試描述項的清單。

每個測試描述項都使用測試群組 ID 和一或多個測試案例 IDs，來識別要從特定測試群組執行的個別測試。測試描述項使用下列格式：

```
GroupId: group-id  
CaseIds: [test-id, test-id] # optional
```

Example範例

下列範例使用您可以定義為的一般內容表達式Aliases。

```
ConditionalTests:  
- Condition: "{{$aliases.Condition1}}"  
  Tests:  
    - GroupId: A  
    - GroupId: B  
- Condition: "{{$aliases.Condition2}}"  
  Tests:  
    - GroupId: D  
- Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"  
  Tests:  
    - GroupId: C
```

根據定義的條件，IDT 會選取測試群組，如下所示：

- 如果 Condition1 為 true，IDT 會在測試群組 A、B 和 C 中執行測試。
- 如果 Condition2 為 true，IDT 會在測試群組 C 和 D 中執行測試。

Order

選用。執行測試的順序。您可以在測試群組層級指定測試順序。如果您未指定本節，IDT 會以隨機順序執行所有適用的測試群組。的值Order是群組描述項清單的清單。您在 中未列出的任何測試群組Order，都可以與任何其他列出的測試群組平行執行。

每個群組描述項清單都包含多個群組描述項的其中一個，並識別執行每個描述項中指定群組的順序。您可以使用下列格式來定義個別群組描述項：

- *group-id* - 現有測試群組的群組 ID。
- [*group-id*, *group-id*] - 可以相對於彼此以任何順序執行的測試群組清單。
- "*"—Wildcard。這相當於目前群組描述項清單中尚未指定的所有測試群組清單。

的值Order也必須符合下列要求：

- 您在群組描述項中指定的測試群組 IDs，必須存在於您的測試套件中。
- 每個群組描述項清單必須至少包含一個測試群組。
- 每個群組描述項清單必須包含唯一的群組 IDs。您無法在個別群組描述項中重複測試群組 ID。
- 群組描述項清單最多可以有一個萬用字元群組描述項。萬用字元群組描述項必須是清單中的第一個或最後一個項目。

Example範例

對於包含測試群組 A、B、C、D 和 E 的測試套件，以下範例清單顯示指定 IDT 應先執行測試群組 A、接著執行測試群組 B，然後以任何順序執行測試群組 C、D 和 E 的不同方式。

- ```
Order:
 - A
 - B
 - [C, D, E]
```

- ```
Order:
  - A
  - B
  - ""
```

- Order:
 - - A
 - B
 - - B
 - C
 - - B
 - D
 - - B
 - E

Features

選用。您希望 IDT 新增至 `awsiotdevicetester_report.xml` 檔案的產品功能清單。如果您未指定此區段，IDT 不會將任何產品功能新增至報告。

產品功能是有關裝置可能符合的特定條件的使用者定義資訊。例如，MQTT 產品功能可以指定裝置正確發佈 MQTT 訊息。在 `awsiotdevicetester_report.xml`，產品功能會根據是否通過指定的測試，設定為 `supported`、`not-supported` 或自訂使用者定義值。

Features 清單中的每個項目都包含下列參數：

Name

功能的名稱。

Value

選用。您想要在報告中使用的自訂值，而不是 `supported`。如果未指定此值，則根據 IDT `not-supported` 會根據測試結果，將功能值設定為 `supported` 或 `not-supported`。如果您使用不同的條件來測試相同的功能，您可以針對 Features 清單中該功能的每個執行個體使用自訂值，IDT 會串連支援條件的功能值。如需詳細資訊，請參閱

Condition

評估為布林值的內容表達式。如果評估的值為 `true`，IDT 會在測試報告完成執行測試套件後，將功能新增至測試報告。如果評估的值為 `false`，則測試不會包含在報告中。

Tests

選用。測試描述項的清單。此清單中指定的所有測試都必須通過，才能支援此功能。

此清單中的每個測試描述項都使用測試群組 ID 和一或多個測試案例 IDs，來識別要從特定測試群組執行的個別測試。測試描述項使用下列格式：

```
GroupId: group-id  
CaseIds: [test-id, test-id] # optional
```

您必須 `OneOfTests` 為 `Features` 清單中的每個功能指定 `Tests` 或。

OneOfTests

選用。測試描述項的清單。此清單中指定的至少一個測試必須通過，才能支援此功能。

此清單中的每個測試描述項都使用測試群組 ID 和一或多個測試案例 IDs，來識別要從特定測試群組執行的個別測試。測試描述項使用下列格式：

```
GroupId: group-id  
CaseIds: [test-id, test-id] # optional
```

您必須 `OneOfTests` 為 `Features` 清單中的每個功能指定 `Tests` 或。

IsRequired

布林值，定義測試報告中是否需要此功能。預設值為 `false`。

Example

測試協調器內容

測試協調程式內容是唯讀 JSON 文件，其中包含在執行期間可供測試協調器使用的資料。測試協調器內容只能從測試協調器存取，並包含決定測試流程的資訊。例如，您可以使用 `userdata.json` 檔案中測試執行器設定的資訊來判斷是否需要執行特定測試。

測試協調器內容使用下列格式：

```
{  
  "pool": {  
    <device-json-pool-element>  
  },  
  "userData": {  
    <userdata-json-content>  
  },  
}
```

```
"config": {  
    <config-json-content>  
}  
}
```

pool

為測試執行選取的裝置集區資訊。對於選取的裝置集區，此資訊會從device.json檔案中定義的對應頂層裝置集區陣列元素擷取。

userData

userdata.json 檔案中的資訊。

config

config.json 檔案中的資訊。

您可以使用 JSONPath 表示法查詢內容。狀態定義中 JSONPath 查詢的語法為 `{{query}}`。當您從測試協調器內容存取資料時，請確定每個值評估為字串、數字或布林值。

如需使用 JSONPath 表示法從內容存取資料的詳細資訊，請參閱 [使用 IDT 內容](#)。

設定 IDT 狀態機器

Important

從 IDT v4.5.1 開始，此狀態機器已棄用。我們強烈建議您使用新的測試協調器。如需詳細資訊，請參閱[設定 IDT 測試協調器](#)。

狀態機器是一種控制測試套件執行流程的建構。它決定測試套件的開始狀態、根據使用者定義的規則管理狀態轉換，並繼續轉換這些狀態，直到達到結束狀態為止。

如果您的測試套件不包含使用者定義的狀態機器，IDT 會為您產生狀態機器。預設狀態機器會執行下列函數：

- 讓測試執行器能夠選取和執行特定測試群組，而不是整個測試套件。
- 如果未選取特定測試群組，會以隨機順序執行測試套件中的每個測試群組。
- 產生報告並列印主控台摘要，顯示每個測試群組和測試案例的測試結果。

IDT 測試套件的狀態機器必須符合下列條件：

- 每個狀態對應於 IDT 要採取的動作，例如執行測試群組或製作報告檔案的產品。
- 轉換為 狀態會執行與 狀態相關聯的動作。
- 每個狀態都會定義下一個狀態的轉換規則。
- 結束狀態必須為 Succeed或 Fail。

狀態機器格式

您可以使用下列範本來設定自己的 `<custom-test-suite-folder>/suite/state_machine.json` 檔案：

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

如下所述，包含值的所有欄位皆為必要：

Comment

狀態機器的描述。

StartAt

IDT 開始執行測試套件的狀態名稱。的值StartAt必須設定為 States 物件中列出的其中一個狀態。

States

將使用者定義狀態名稱映射至有效 IDT 狀態的物件。每個 States.*state-name* 物件都包含映射至 *state-name* 的有效狀態定義。

States 物件必須包含 Succeed 和 Fail 狀態。如需有效狀態的資訊，請參閱 [有效狀態和狀態定義](#)。

有效狀態和狀態定義

本節說明可在 IDT 狀態機器中使用的所有有效狀態的狀態定義。下列某些狀態支援測試案例層級的組態。不過，建議您在測試群組層級設定狀態轉換規則，而非測試案例層級，除非絕對必要。

狀態定義

- [RunTask](#)
- [Choice](#)
- [平行](#)
- [AddProductFeatures](#)
- [報告](#)
- [LogMessage](#)
- [SelectGroup](#)
- [失敗](#)
- [Succeed](#)

RunTask

RunTask 狀態會從測試套件中定義的測試群組執行測試案例。

```
{
  "Type": "RunTask",
  "Next": "<state-name>",
  "TestGroup": "<group-id>",
  "TestCases": [
    "<test-id>"
  ],
  "ResultVar": "<result-name>"
}
```


如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

TestGroup

選用。要執行的測試群組 ID。如果未指定此值，IDT 會執行測試執行器選取的測試群組。

TestCases

選用。來自 中指定群組的測試案例 IDs 陣列TestGroup。根據 TestGroup和 的值TestCases，IDT 會決定測試執行行為，如下所示：

- 同時指定 TestCases TestGroup和 時，IDT 會從測試群組執行指定的測試案例。
- TestCases 指定TestGroup但未指定 時，IDT 會執行指定的測試案例。
- 指定 TestCases 但未指定 TestGroup 時，IDT 會執行指定測試群組中的所有測試案例。
- 未指定 TestCases TestGroup或 時，IDT 會從測試執行器從 IDT CLI 中選取的測試群組執行所有測試案例。若要啟用測試執行器的群組選擇，您必須在 state_machine.json 檔案中同時包含 RunTask和 Choice 狀態。如需如何運作的範例，請參閱[範例狀態機器：執行使用者選取的測試群組](#)。

如需為測試執行器啟用 IDT CLI 命令的詳細資訊，請參閱 [the section called “啟用 IDT CLI 命令”](#)。

ResultVar

使用測試執行結果設定的內容變數名稱。如果您未指定 的值，請勿指定此值TestGroup。IDT false會根據下列項目，將您在 中定義的變數值設定為 ResultVartrue或：

- 如果變數名稱的格式為 `text_text_passed`，則值會設定為是否通過或略過第一個測試群組中的所有測試。
- 在所有其他情況下，該值會設定為是否通過或略過所有測試群組中的所有測試。

一般而言，您將使用 RunTask 狀態來指定測試群組 ID，而不指定個別測試案例 IDs，以便 IDT 執行指定測試群組中的所有測試案例。此狀態執行的所有測試案例都會以隨機順序平行執行。不過，如果所有測試案例都需要裝置才能執行，而且只有單一裝置可用，則測試案例將改為依序執行。

錯誤處理

如果任何指定的測試群組或測試案例 IDs 無效，則此狀態會發出RunTaskError執行錯誤。如果狀態遇到執行錯誤，則也會將狀態機器內容中的hasExecutionError變數設定為 true。

Choice

Choice 狀態可讓您動態設定下一個狀態，以根據使用者定義的條件轉換為。

```
{
  "Type": "Choice",
  "Default": "<state-name>",
  "FallthroughOnError": true | false,
  "Choices": [
    {
      "Expression": "<expression>",
      "Next": "<state-name>"
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

Default

如果 中定義的任何表達式Choices都無法評估為 `true`，則要轉換為的預設狀態`true`。

FallthroughOnError

選用。指定狀態在評估表達式時發生錯誤時的行為。`true` 如果您想要在評估導致錯誤時略過表達式，請將 設定為 `true`。如果沒有相符的表達式，則狀態機器會轉換為 Default 狀態。如果未指定FallthroughOnError值，則預設為 `false`。

Choices

運算式和狀態的陣列，用於判斷在目前狀態下執行動作後要轉換到哪個狀態。

Choices.Expression

評估為布林值的表達式字串。如果表達式評估為 `true`，則狀態機器會轉換為 中定義的狀態Choices.Next。表達式字串會從狀態機器內容擷取值，然後對其執行操作以到達布林值。如需存取狀態機器內容的資訊，請參閱 [狀態機器內容](#)。

Choices.Next

如果 中定義的表達式Choices.Expression評估為 `true`，則要轉換為的狀態名稱`true`。

錯誤處理

在下列情況下，Choice 狀態可能需要錯誤處理：

- 選擇表達式中的某些變數不存在於狀態機器內容中。
- 表達式的結果不是布林值。
- JSON 查詢的結果不是字串、數字或布林值。

您無法使用Catch區塊來處理此狀態的錯誤。如果您想要在遇到錯誤時停止執行狀態機器，您必須將FallthroughOnError設定為 false。不過，我們建議您將FallthroughOnError設定為 true，並根據您的使用案例執行下列其中一項操作：

- 如果您存取的變數預期在某些情況下不存在，請使用 值Default和其他Choices區塊來指定下一個狀態。
- 如果您存取的變數應一律存在，請將Default狀態設定為 Fail。

平行

Parallel 狀態可讓您彼此平行定義和執行新的狀態機器。

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Branches": [
    <state-machine-definition>
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

Branches

要執行的狀態機器定義陣列。每個狀態機器定義都必須包含自己的 StartAt、Succeed和 Fail 狀態。此陣列中的狀態機器定義無法參考其定義以外的狀態。

Note

由於每個分支狀態機器共用相同的狀態機器內容，因此在一個分支中設定變數，然後從另一個分支讀取這些變數可能會導致意外行為。

Parallel 狀態只有在執行所有分支狀態機器之後才會移至下一個狀態。需要裝置的每個狀態都會等待執行，直到裝置可用為止。如果有多個裝置可用，此狀態會從多個群組平行執行測試案例。如果沒有足夠的裝置可用，則測試案例將依序執行。由於測試案例在平行執行時以隨機順序執行，因此可能會使用不同的裝置從相同的測試群組執行測試。

錯誤處理

請確定分支狀態機器和父系狀態機器都會轉換為 Fail 狀態，以處理執行錯誤。

由於分支狀態機器不會將執行錯誤傳輸至父系狀態機器，因此您無法使用Catch區塊來處理分支狀態機器中的執行錯誤。反之，請在共用狀態機器內容中使用 hasExecutionErrors 值。如需如何運作的範例，請參閱 [狀態機器範例：平行執行兩個測試群組](#)。

AddProductFeatures

AddProductFeatures 狀態可讓您將產品功能新增至 IDT 產生的awsiotdevicetester_report.xml檔案。

產品功能是有關裝置可能符合的特定條件的使用者定義資訊。例如，MQTT產品功能可以指定裝置正確發佈 MQTT 訊息。在報告中，產品功能會根據是否通過指定的測試，設定為 supported、not-supported或自訂值。

Note

AddProductFeatures 狀態本身不會產生報告。此狀態必須轉換為 [Report 狀態](#) 才能產生報告。

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Features": [
    {
      "Feature": "<feature-name>",
      "Groups": [
        "<group-id>"
      ],
      "OneOfGroups": [
        "<group-id>"
      ],
    }
  ],
}
```

```

        "TestCases": [
            "<test-id>"
        ],
        "IsRequired": true | false,
        "ExecutionMethods": [
            "<execution-method>"
        ]
    }
}

```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

Features

要在 `awsiotdevicetester_report.xml` 檔案中顯示的產品功能陣列。

Feature

功能的名稱

FeatureValue

選用。要在報告中使用的自訂值，而非 `supported`。如果未指定此值，則根據測試結果，功能值會設為 `supported` 或 `not-supported`。

如果您使用的自訂值 `FeatureValue`，則可以使用不同的條件測試相同的功能，IDT 會串連支援條件的功能值。例如，以下摘錄顯示具有兩個不同 `MyFeature` 特徵值的功能：

```

...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},

```

...

如果兩個測試群組都通過，則功能值會設為 `first-feature-supported`，`second-feature-supported`。

Groups

選用。測試群組 IDs 的陣列。每個指定測試群組中的所有測試都必須通過，才能支援此功能。

OneOfGroups

選用。測試群組 IDs 的陣列。至少一個指定測試群組內的所有測試都必須通過，才能支援此功能。

TestCases

選用。測試案例 IDs 的陣列。如果您指定此值，則適用下列條件：

- 所有指定的測試案例都必須通過，才能支援此功能。
- Groups 只能包含一個測試群組 ID。
- OneOfGroups 不得指定。

IsRequired

選用。設定為 `false` 以在報告中將此功能標記為選用功能。預設值為 `true`。

ExecutionMethods

選用。符合 `device.json` 檔案中指定 `protocol` 值的執行方法陣列。如果指定此值，則測試執行器必須指定 `protocol` 符合此陣列中其中一個值的值，以在報告中包含 功能。如果未指定此值，則功能一律會包含在報告中。

若要使用 `AddProductFeatures` 狀態，您必須將 `RunTask` 狀態 `ResultVar` 中的 值設定為下列其中一個值：

- 如果您指定個別測試案例 IDs，請將 `ResultVar` 設定為 `group-id_test-id_passed`。
- 如果您未指定個別測試案例 IDs，請將 `ResultVar` 設定為 `group-id_passed`。

`AddProductFeatures` 狀態會以下列方式檢查測試結果：

- 如果您未指定任何測試案例 IDs，則每個測試群組的結果取決於狀態機器內容中的 `group-id_passed` 變數值。

- 如果您確實指定測試案例 IDs，則每個測試的結果取決於狀態機器內容中的 *group-id_test-id_passed* 變數值。

錯誤處理

如果在此狀態下提供的群組 ID 不是有效的群組 ID，則此狀態會導致 `AddProductFeaturesError` 執行錯誤。如果狀態遇到執行錯誤，則也會將狀態機器內容中的 `hasExecutionErrors` 變數設定為 `true`。

報告

Report 狀態會產生 *suite-name*_Report.xml 和 `awsiotdevicetester_report.xml` 檔案。此狀態也會將報告串流至 主控台。

```
{
  "Type": "Report",
  "Next": "<state-name>"
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

您應該一律在測試執行流程結束時轉換為 Report 狀態，以便測試執行器可以檢視測試結果。一般而言，此狀態之後的下一個狀態為 Succeed。

錯誤處理

如果此狀態在產生報告時遇到問題，則會發出 `ReportError` 執行錯誤。

LogMessage

LogMessage 狀態會產生 `test_manager.log` 檔案，並將日誌訊息串流至 主控台。

```
{
  "Type": "LogMessage",
  "Next": "<state-name>"
  "Level": "info | warn | error"
  "Message": "<message>"
}
```

```
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

Level

建立日誌訊息的錯誤層級。如果您指定的關卡無效，此狀態會產生錯誤訊息並予以捨棄。

Message

要記錄的訊息。

SelectGroup

SelectGroup 狀態會更新狀態機器內容，以指出已選取哪些群組。此狀態設定的值會由任何後續Choice狀態使用。

```
{
  "Type": "SelectGroup",
  "Next": "<state-name>"
  "TestGroups": [
    <group-id>"
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

Next

在目前狀態下執行動作後，要轉換為的狀態名稱。

TestGroups

將標記為選取的測試群組陣列。對於此陣列中的每個測試群組 ID，*group-id_selected*變數會在內容true中設定為。請務必提供有效的測試群組 IDs因為 IDT 不會驗證指定的群組是否存在。

失敗

Fail 狀態表示狀態機器未正確執行。這是狀態機器的結束狀態，每個狀態機器定義都必須包含此狀態。


```
{
  "Type": "Fail"
}
```

Succeed

Succeed 狀態表示狀態機器已正確執行。這是狀態機器的結束狀態，每個狀態機器定義都必須包含此狀態。

```
{
  "Type": "Succeed"
}
```

狀態機器內容

狀態機器內容是唯讀 JSON 文件，其中包含可在執行期間提供給狀態機器的資料。狀態機器內容只能從狀態機器存取，並包含決定測試流程的資訊。例如，您可以使用 `userdata.json` 檔案中測試執行器設定的資訊來判斷是否需要執行特定測試。

狀態機器內容使用下列格式：

```
{
  "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

為測試執行選取之裝置集區的相關資訊。對於選取的裝置集區，此資訊會從 `device.json` 檔案中定義的對應頂層裝置集區陣列元素擷取。

userData

`userdata.json` 檔案中的資訊。

config

資訊鎖定 `config.json` 檔案。

suiteFailed

狀態機器啟動 `false` 時，此值會設為 `false`。如果測試群組在 `RunTask` 狀態失敗，則此值會在狀態機器執行的剩餘持續時間內設定為 `true`。

specificTestGroups

如果測試執行器選取要執行的特定測試群組，而不是整個測試套件，則會建立此金鑰並包含特定測試群組 IDs 的清單。

specificTestCases

如果測試執行器選取要執行的特定測試案例，而不是整個測試套件，則會建立此金鑰並包含特定測試案例 IDs 的清單。

hasExecutionErrors

狀態機器啟動時不會結束。如果任何狀態遇到執行錯誤，則會建立此變數 `true`，並在狀態機器執行的剩餘持續時間內設定為 `true`。

您可以使用 JSONPath 表示法查詢內容。狀態定義中 JSONPath 查詢的語法為 `{{$.query}}`。您可以在某些狀態下使用 JSONPath 查詢做為預留位置字串。IDT 會將預留位置字串取代為內容中評估 JSONPath 查詢的值。您可以針對下列值使用預留位置：

- 狀態 `TestCases` 的值 `RunTask`。
- `Expression` 值 `Choice` 狀態。

當您從狀態機器內容存取資料時，請確定符合下列條件：

- 您的 JSON 路徑必須以開頭 `$.`
- 每個值都必須評估為字串、數字或布林值。

如需使用 JSONPath 表示法從內容存取資料的詳細資訊，請參閱 [使用 IDT 內容](#)。

執行錯誤

執行錯誤是狀態機器在執行狀態時遇到的狀態機器定義錯誤。IDT 會記錄 `test_manager.log` 檔案中每個錯誤的相關資訊，並將日誌訊息串流至主控台。

您可以使用下列方法來處理執行錯誤：

- 在狀態定義中新增 [Catch 區塊](#)。
- 在狀態機器內容中檢查 [hasExecutionErrors](#) 值的值。

擷取

若要使用 Catch，請將下列項目新增至您的狀態定義：

```
"Catch": [  
  {  
    "ErrorEquals": [  
      "<error-type>"  
    ]  
    "Next": "<state-name>"  
  }  
]
```

如下所述，包含值的所有欄位皆為必要：

Catch.ErrorEquals

要擷取的錯誤類型陣列。如果執行錯誤符合其中一個指定的值，則狀態機器會轉換為 中指定的狀態 `Catch.Next`。如需其產生的錯誤類型的相關資訊，請參閱每個狀態定義。

Catch.Next

如果目前狀態遇到符合 中指定其中一個值的執行錯誤，則會轉換為下一個狀態 `Catch.ErrorEquals`。

擷取區塊會依序處理，直到一個相符為止。如果沒有錯誤符合 Catch 區塊中列出的錯誤，則狀態機器會繼續執行。由於執行錯誤是不正確狀態定義的結果，我們建議您在狀態遇到執行錯誤時轉換為失敗狀態。

hasExecutionError

當某些狀態遇到執行錯誤時，除了發出錯誤之外，也會在狀態機器內容 `true` 中將 `hasExecutionError` 值設定為 `true`。您可以使用此值來偵測何時發生錯誤，然後使用 `Choice` 狀態將狀態機器轉換為 `Fail` 狀態。

此方法具有下列特性。

- 狀態機器不會以指派給的任何值開始 `hasExecutionError`，且此值在特定狀態設定之前無法使用。這表示您必須 `false` 針對存取此值 `Choice` 的狀態，明確 `FallthroughOnError` 將設定為 `true`，以防止狀態機器在未發生執行錯誤時停止。
- 一旦設定為 `true`，`hasExecutionError` 永遠不會設定為 `false` 或從內容中移除。這表示此值只有在第一次設定為 `true` 時才有用 `true`，而且對於所有後續狀態，它不提供有意義的值。
- 該 `hasExecutionError` 值會與 `Parallel` 狀態的所有分支狀態機器共用，這可能會導致非預期的結果，具體取決於存取它的順序。

由於這些特性，如果您可以改用 `Catch` 區塊，我們不建議您使用此方法。

狀態機器範例

本節提供一些範例狀態機器組態。

範例

- [狀態機器範例：執行單一測試群組](#)
- [狀態機器範例：執行使用者選取的測試群組](#)
- [狀態機器範例：使用產品功能執行單一測試群組](#)
- [狀態機器範例：平行執行兩個測試群組](#)

狀態機器範例：執行單一測試群組

此狀態機器：

- 執行 ID 為 `GroupA` 的測試群組 `GroupA`，其必須存在於 `group.json` 檔案的套件中。
- 檢查執行錯誤，並在發現任何 `Fail` 時轉換為 `Fail`。
- `Succeed` 如果沒有錯誤，則產生報告並轉換為 `Succeed`，`Fail` 否則。

```
{
```

```
"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"
  }
}
```

狀態機器範例：執行使用者選取的測試群組

此狀態機器：

- 檢查測試執行器是否已選取特定測試群組。狀態機器不會檢查特定測試案例，因為測試執行器無法在未同時選取測試群組的情況下選取測試案例。

- 如果選取測試群組：
 - 在選取的測試群組中執行測試案例。若要這樣做，狀態機器不會明確指定 RunTask 狀態中的任何測試群組或測試案例。
 - 執行所有測試並結束之後產生報告。
- 如果未選取測試群組：
 - 在測試群組 中執行測試GroupA。
 - 產生報告並結束。

```
{
  "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"
    }
}
}

```

狀態機器範例：使用產品功能執行單一測試群組

此狀態機器：

- 執行測試群組 GroupA。
- 檢查執行錯誤，並在發現任何 Fail 時轉換為 。
- 將 FeatureThatDependsOnGroupA 功能新增至 awsiotdevicetester_report.xml 檔案：
 - 如果 GroupA 通過，則功能會設定為 supported。
 - 該功能在報告中不會標記為選用。
- Succeed 如果沒有錯誤，則產生報告並轉換為 ，Fail 否則

```
{
  "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"
    }
  }
}

```

狀態機器範例：平行執行兩個測試群組

此狀態機器：

- 平行執行 GroupA 和 GroupB 測試群組。分支狀態機器中 RunTask 狀態存放在內容中的 ResultVar 變數可供 AddProductFeatures 狀態使用。
- 檢查執行錯誤，並在發現任何 Fail 時轉換為 。此狀態機器不會使用 Catch 區塊，因為該方法不會偵測分支狀態機器中的執行錯誤。
- 根據傳遞的群組，將功能新增至 awsiotdevicetester_report.xml 檔案
 - 如果 GroupA 通過，則功能會設定為 supported。
 - 該功能在報告中不會標記為選用。
- Succeed 如果沒有錯誤，則產生報告並轉換為 ，Fail 否則

如果在裝置集區中設定了兩個裝置，則 GroupA 和 GroupB 可以同時執行。不過，如果 GroupA 或有 GroupB 多個測試，則兩個裝置可能會配置到這些測試。如果只設定一個裝置，測試群組將依序執行。

```

{
  "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"
    }
  }
}
```

建立 IDT 測試案例可執行檔

您可以透過下列方式，在測試套件資料夾中建立和放置測試案例可執行檔：

- 對於使用 `test.json` 檔案中的引數或環境變數來判斷要執行哪些測試的測試套件，您可以建立整個測試套件的單一測試案例可執行檔，或測試套件中每個測試群組的測試可執行檔。
- 對於您想要根據指定命令執行特定測試的測試套件，您可以在測試套件中為每個測試案例建立一個可執行的測試案例。

身為測試寫入器，您可以判斷哪種方法適合您的使用案例，並據以建構您的測試案例可執行檔。請確定您在每個 `test.json` 檔案中提供正確的測試案例可執行路徑，且指定的可執行檔正確執行。

當所有裝置準備好執行測試案例時，IDT 會讀取下列檔案：

- 所選測試案例 `test.json` 的 會決定要啟動的程序，以及要設定的環境變數。
- 測試套件 `suite.json` 的 決定要設定的環境變數。

IDT 會根據 `test.json` 檔案中指定的命令和引數啟動必要的測試可執行程序，並將所需的環境變數傳遞給程序。

使用 IDT 用戶端 SDK

IDT 用戶端 SDKs 可讓您使用 API 命令，簡化在測試可執行檔中編寫測試邏輯的方式，您可以使用該命令與 IDT 和受測裝置互動。IDT 目前提供下列 SDKs：

- 適用於 Python 的 IDT 用戶端 SDK
- IDT Client SDK for Go

- 適用於 Java 的 IDT 用戶端 SDK

這些 SDKs 位於 *<device-tester-extract-location>/sdks* 資料夾中。建立新的測試案例可執行檔時，您必須將要使用的 SDK 複製到包含測試案例可執行檔的資料夾，並在程式碼中參考 SDK。本節提供可在測試案例可執行檔中使用的可用 API 命令的簡短描述。

本節內容

- [裝置互動](#)
- [IDT 互動](#)
- [主機互動](#)

裝置互動

下列命令可讓您與受測裝置通訊，而不必實作任何其他裝置互動和連線管理功能。

ExecuteOnDevice

允許測試套件在支援 SSH 或 Docker shell 連線的裝置上執行 Shell 命令。

CopyToDevice

允許測試套件將本機檔案從執行 IDT 的主機機器複製到支援 SSH 或 Docker shell 連線之裝置上的指定位置。

ReadFromDevice

允許測試套件從支援 UART 連線的裝置序列連接埠讀取。

Note

由於 IDT 不會管理使用內容中的裝置存取資訊所建立之裝置的直接連線，我們建議您在測試案例可執行檔中使用這些裝置互動 API 命令。不過，如果這些命令不符合您的測試案例要求，則您可以從 IDT 內容擷取裝置存取資訊，並使用它從測試套件直接連線至裝置。

若要進行直接連線，請分別擷取 `device.connectivity` 以及受測裝置和資源裝置 `resource.devices.connectivity` 的欄位。如需使用 IDT 內容的詳細資訊，請參閱 [使用 IDT 內容](#)。

IDT 互動

下列命令可讓您的測試套件與 IDT 通訊。

PollForNotifications

允許測試套件檢查來自 IDT 的通知。

GetContextValue 和 GetContextString

允許測試套件從 IDT 內容擷取值。如需詳細資訊，請參閱[使用 IDT 內容](#)。

SendResult

允許測試套件向 IDT 報告測試案例結果。此命令必須在測試套件中每個測試案例的結尾呼叫。

主機互動

下列命令可讓您的測試套件與主機機器通訊。

PollForNotifications

允許測試套件檢查來自 IDT 的通知。

GetContextValue 和 GetContextString

允許測試套件從 IDT 內容擷取值。如需詳細資訊，請參閱[使用 IDT 內容](#)。

ExecuteOnHost

允許測試套件在本機機器上執行命令，並讓 IDT 管理測試案例可執行生命週期。

啟用 IDT CLI 命令

run-suite 命令 IDT CLI 提供多種選項，可讓測試執行器自訂測試執行。若要允許測試執行器使用這些選項來執行自訂測試套件，請實作對 IDT CLI 的支援。如果您未實作支援，測試執行器仍然可以執行測試，但某些 CLI 選項將無法正常運作。為了提供理想的客戶體驗，建議您在 IDT CLI 中為 run-suite 命令實作下列引數的支援：

timeout-multiplier

指定大於 1.0 的值，該值將在執行測試時套用至所有逾時。

測試執行器可以使用此引數來增加他們想要執行的測試案例的逾時。當測試執行器在其run-suite命令中指定此引數時，IDT 會使用它來計算 IDT_TEST_TIMEOUT 環境變數的值，並在 IDT 內容中設定 config.timeoutMultiplier 欄位。若要支援此引數，您必須執行下列動作：

- 讀取 IDT_TEST_TIMEOUT 環境變數以取得正確計算的逾時值，而不是直接使用 test.json 檔案的逾時值。
- 從 IDT 內容擷取config.timeoutMultiplier值，並將其套用至長時間執行的逾時。

如需因逾時事件而提早結束的詳細資訊，請參閱[指定結束行為](#)。

stop-on-first-failure

指定如果 IDT 遇到失敗，應停止執行所有測試。

當測試執行器在其run-suite命令中指定此引數時，IDT 會在遇到失敗時立即停止執行測試。不過，如果測試案例平行執行，則這可能會導致非預期的結果。若要實作支援，請確定如果 IDT 遇到此事件，您的測試邏輯會指示所有執行中的測試案例停止、清除臨時資源，並向 IDT 報告測試結果。如需在故障時提早結束的詳細資訊，請參閱 [指定結束行為](#)。

group-id 和 test-id

指定 IDT 應僅執行選取的測試群組或測試案例。

測試執行器可以將這些引數與其run-suite命令搭配使用，以指定下列測試執行行為：

- 在指定的測試群組內執行所有測試。
- 在指定的測試群組內執行一系列的測試。

若要支援這些引數，測試套件的測試協調器必須在測試協調器中包含一組特定的 RunTask和 Choice 狀態。如果您不使用自訂狀態機器，則預設的 IDT 測試協調器會包含您所需的狀態，而且您不需要採取其他動作。不過，如果您使用的是自訂測試協調器，則使用 [狀態機器範例：執行使用者選取的測試群組](#)做為範例，在您的測試協調器中新增所需的狀態。

如需 IDT CLI 命令的詳細資訊，請參閱 [偵錯並執行自訂測試套件](#)。

寫入事件日誌

測試執行時，您可以將資料傳送至 stdout和 stderr，以將事件日誌和錯誤訊息寫入主控台。如需主控台訊息格式的相關資訊，請參閱[主控台訊息格式](#)。

當 IDT 完成執行測試套件時，此資訊也可在 資料夾的 test_manager.log 檔案中取得<devicetester-extract-location>/results/<execution-id>/logs。

您可以設定每個測試案例，將日誌從其測試執行寫入至位於 `<device-tester-extract-location>/results/execution-id/logs` 資料夾中 `<group-id>_<test-id>` 的檔案，包括測試中裝置的日誌。若要執行此操作，請從 IDT 內容中擷取日誌檔案的路徑 `testData.logFilePath`，在該路徑建立檔案，然後寫入您想要的內容。IDT 會根據正在執行的測試案例自動更新路徑。如果您選擇不建立測試案例的日誌檔案，則不會產生該測試案例的檔案。

您也可以設定文字可執行檔，視需要在 `<device-tester-extract-location>/logs` 資料夾中建立其他日誌檔案。建議您為日誌檔案名稱指定唯一的字首，以免您的檔案遭到覆寫。

向 IDT 報告結果

IDT 會將測試結果寫入 `awsiotdevicetester_report.xml` 和 `suite-name_report.xml` 檔案。這些報告檔案位於 `<device-tester-extract-location>/results/<execution-id>/` 兩個報告都會擷取測試套件執行的結果。如需 IDT 用於這些報告之結構描述的詳細資訊，請參閱 [檢閱 IDT 測試結果和日誌](#)

若要填入 `suite-name_report.xml` 檔案的內容，您必須在測試執行完成之前使用 `SendResult` 命令向 IDT 報告測試結果。如果 IDT 找不到測試的結果，則會發出測試案例的錯誤。下列 Python 摘錄顯示將測試結果傳送至 IDT 的命令：

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

如果您未透過 API 報告結果，IDT 會在測試成品資料夾中尋找測試結果。此資料夾的路徑會存放在 IDT 內容中 `testData.testArtifactsPath` 存檔的中。在此資料夾中，IDT 會使用第一個依字母順序排序的 XML 檔案做為測試結果。

如果您的測試邏輯產生 JUnit XML 結果，您可以將測試結果寫入成品資料夾中的 XML 檔案，以直接將結果提供給 IDT，而不是剖析結果，然後使用 API 將結果提交給 IDT。

如果您使用此方法，請確定您的測試邏輯準確摘要測試結果，並以與檔案相同的格式格式化結果 `suite-name_report.xml` 檔案。IDT 不會對您提供的資料執行任何驗證，但以下情況除外：

- IDT 會忽略 `testsuites` 標籤的所有屬性。反之，它會從其他報告的測試群組結果計算標籤屬性。
- 必須至少有一個 `testsuite` 標籤存在於 `testsuites` 中。

由於 IDT 在所有測試案例使用相同的成品資料夾，且不會在測試執行之間刪除結果檔案，因此如果 IDT 讀取不正確的檔案，此方法也可能會導致錯誤報告。建議您在所有測試案例使用與產生 XML 結果

檔案相同的名稱，以覆寫每個測試案例的結果，並確保 IDT 可以使用正確的結果。雖然您可以使用混合方法在測試套件中報告，也就是說，對於某些測試案例使用 XML 結果檔案，並透過 API 為其他人提交結果，但不建議使用此方法。

指定結束行為

將您的文字可執行檔設定為一律以結束碼 0 結束，即使測試案例報告失敗或錯誤結果亦然。僅使用非零結束碼來表示測試案例未執行，或測試案例可執行檔無法將任何結果傳達給 IDT。當 IDT 收到非零結束碼時，它會標記測試案例發生錯誤，導致無法執行。

IDT 可能會請求或預期測試案例在下列事件完成之前停止執行。使用此資訊來設定您的測試案例可執行檔，以偵測測試案例中的每個事件：

Timeout (逾時)

當測試案例執行的時間超過 `test.json` 檔案中指定的逾時值時，便會發生。如果測試執行器使用 `timeout-multiplier` 引數指定逾時乘數，則 IDT 會使用乘數計算逾時值。

若要偵測此事件，請使用 `IDT_TEST_TIMEOUT` 環境變數。當測試執行器啟動測試時，IDT 會將 `IDT_TEST_TIMEOUT` 環境變數的值設定為計算的逾時值（以秒為單位），並將變數傳遞至測試案例可執行檔。您可以讀取變數值來設定適當的計時器。

中斷

當測試執行器中斷 IDT 時發生。例如，按 `Ctrl+C`。

由於終端機會將訊號傳播到所有子程序，因此您只需在測試案例中設定訊號處理常式，即可偵測中斷訊號。

或者，您可以定期輪詢 API，以檢查 `PollForNotifications API CancellationRequested` 回應中的布林值。當 IDT `CancellationRequested` 收到中斷訊號時，它會將布林值設定為 `true`。

第一次失敗時停止

當與目前測試案例平行執行的測試案例失敗，且測試執行器使用 `stop-on-first-failure` 引數指定 IDT 應在遇到任何失敗時停止時，便會發生。

若要偵測此事件，您可以定期輪詢 API，以檢查 `PollForNotifications API CancellationRequested` 回應中的布林值。當 IDT 遇到故障，且設定為在第一次故障時停止時，它會將 `CancellationRequested` 布林值設定為 `true`。

當這些事件發生時，IDT 會等待 5 分鐘，讓任何目前正在執行的測試案例完成執行。如果所有執行中的測試案例未在 5 分鐘內結束，IDT 會強制其每個程序停止。如果 IDT 未在程序結束前收到測試結果，則會將測試案例標記為已逾時。最佳實務是，您應該確保測試案例在遇到其中一個事件時執行下列動作：

1. 停止執行正常測試邏輯。
2. 清除任何臨時資源，例如受測裝置上的測試成品。
3. 向 IDT 報告測試結果，例如測試失敗或錯誤。
4. 結束。

使用 IDT 內容

當 IDT 執行測試套件時，測試套件可以存取一組資料，用於判斷每個測試的執行方式。此資料稱為 IDT 內容。例如，由 `userdata.json` 檔案中的測試執行器提供的使用者資料組態，可用於在 IDT 內容中測試套件。

IDT 內容可以視為唯讀 JSON 文件。測試套件可以使用標準 JSON 資料類型，例如物件、陣列、數字等，從擷取資料並將資料寫入內容。

內容結構描述

IDT 內容使用下列格式：

```
{
  "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

來自 [config.json 檔案](#) 的資訊。config 欄位也包含下列其他欄位：

config.timeoutMultiplier

測試套件所使用的任何逾時值的乘數。此值由 IDT CLI 的測試執行器指定。預設值為 1。

device

為測試執行選取的裝置的相關資訊。此資訊等同於所選裝置[device.json 檔案中](#)的devices陣列元素。

devicePool

為測試執行選取之裝置集區的相關資訊。此資訊等同於在所選裝置集區device.json檔案中定義的最上層裝置集區陣列元素。

resource

檔案中資源裝置的相關資訊resource.json。

resource.devices

此資訊等同於 resource.json 檔案中定義的devices陣列。每個devices元素都包含下列其他欄位：

resource.device.name

資源裝置的名稱。此值會設定為 test.json 檔案中的requiredResource.name值。

`testData.awsCredentials`

測試用來連線至 AWS 雲端的 AWS 登入資料。此資訊是從 `config.json` 檔案取得。

`testData.logFilePath`

測試案例寫入日誌訊息的日誌檔案路徑。測試套件會在檔案不存在時建立此檔案。

`userData`

[userdata.json 檔案](#)中測試執行器提供的資訊。

存取內容中的資料

您可以使用 JSON 檔案的 JSONPath 表示法，以及使用 `GetContextValue` 和 `GetContextString` APIs 的可執行文字來查詢內容。用於存取 IDT 內容的 JSONPath 字串語法會有所不同，如下所示：

- 在 `suite.json` 和 `中test.json`，您可以使用 `{{query}}`。也就是說，請勿使用根元素 `$` 來啟動表達式。
- 在 `中test_orchestrator.yaml`，您使用 `{{query}}`。

如果您使用已棄用的狀態機器，則在 `中state_machine.json`，您可以使用 `{{$.query}}`。

- 在 API 命令中，您可以使用 `query` 或 `{{$.query}}`，視命令而定。如需詳細資訊，請參閱 SDKs 中的內嵌文件。

下表說明典型 JSONPath 表達式中的運算子：

Operator	Description
<code>\$</code>	The root element. Because the top-level context value for IDT is an object, you will typically use <code>\$.</code> to start your queries.
<code>.childName</code>	Accesses the child element with name <code>childName</code> 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

Operator	Description
	access the <code>awsRegion</code> value in the config object is <code>\$.config.awsRegion</code> .
<code>#####</code>	Filters elements from an array, retrieving items beginning from the <code>##</code> index and going up to the end index, both inclusive.
<code>#index1# index2#...# indexN#</code>	Filters elements from an array, retrieving items from only the specified indices.
<code>##(expr)#</code>	Filters elements from an array using the <code>expr</code> expression. This expression must evaluate to a boolean value.

若要建立篩選條件表達式，請使用下列語法：

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

在此語法中：

- `jsonpath` 是使用標準 JSON 語法的 JSONPath。
- `value` 是使用標準 JSON 語法的任何自訂值。
- `operator` 是下列其中一個運算子：
 - `<` (小於)
 - `<=` (小於或等於)
 - `==` (等於)

如果表達式中的 JSONPath 或 值是陣列、布林值或物件值，則這是您唯一可以使用的支援二進位運算子。

- `>=` (大於或等於)
- `>` (大於)
- `=~` (規則表達式比對)。若要在篩選條件表達式中使用此運算子，表達式左側的 JSONPath 或值必須評估為字串，而右側必須是遵循 [RE2 語法](#) 的模式值。

您可以使用格式為 `{{query}}` 的 JSONPath 查詢，做為 `test.json` 檔案中 `args` 和 `environmentVariables` 欄位的預留位置字串，以及 `suite.json` 檔案中 `environmentVariables` 的欄位。IDT 會執行內容查詢，並將查詢的評估值填入欄位。例如，在 `suite.json` 檔案中，您可以使用預留位置字串來指定環境變數值，這些值會隨每個測試案例而變更，IDT 會將每個測試案例的正確值填入環境變數。不過，當您在 `test.json` 和 `suite.json` 檔案中使用預留位置字串時，下列考量適用於您的查詢：

- 在所有小情況下，您必須每次在查詢中出現 `devicePool` 金鑰。也就是說，請 `devicepool` 改用。
- 對於陣列，您只能使用字串陣列。此外，陣列使用非標準 `item1, item2, ..., itemN` 格式。如果陣列只包含一個元素，則會將其序列化為 `item`，使其與字串欄位不區分。
- 您無法使用預留位置從內容擷取物件。

由於這些考量，我們建議您盡可能使用 API 來存取測試邏輯中的內容，而不是 `test.json` 和 `suite.json` 檔案中的預留位置字串。不過，在某些情況下，使用 JSONPath 預留位置來擷取單一字串以設定為環境變數可能會更方便。

設定測試執行器的設定

若要執行自訂測試套件，測試執行器必須根據他們想要執行的測試套件來設定其設定。根據位於 `<device-tester-extract-location>/configs/` 資料夾中的組態檔案範本來指定設定。如有需要，測試執行器也必須設定 IDT 用來連線至 AWS 雲端的 AWS 登入資料。

身為測試寫入器，您需要設定這些檔案來[偵錯測試套件](#)。您必須提供測試執行器的說明，以便他們可以視需要設定下列設定來執行您的測試套件。

設定 device.json

`device.json` 檔案包含執行測試之裝置的相關資訊（例如 IP 地址、登入資訊、作業系統和 CPU 架構）。

測試執行器可以使用位於 `<device-tester-extract-location>/configs/` 資料夾的下列範本 `device.json` 檔案提供此資訊。

```
[
  {
    "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>",
    }
]
}
]

```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

sku

可唯一識別測試裝置的英數字元值。SKU 用於追蹤合格裝置。

Note

如果您想要在 AWS Partner Device Catalog 中列出您的電路板，您在此處指定的 SKU 必須符合您在列出程序中使用的 SKU。

features

選用。包含裝置支援功能的陣列。裝置功能是您測試套件中設定的使用者定義值。您必須提供測試執行器有關要包含在 `device.json` 檔案中的功能名稱和值的資訊。例如，如果您想要測試可做為其他裝置的 MQTT 伺服器運作的裝置，則可以設定測試邏輯來驗證名為 `MQTT_QOS` 之功能的特定支援層級 `MQTT_QOS`。測試執行器提供此功能名稱，並將功能值設定為其裝置支援的 QOS 層級。您可以從具有 `devicePool.features` 查詢的 [IDT 內容](#)，或從具有 `pool.features` 查詢的 [測試協調器內容](#) 擷取提供的資訊。

features.name

功能的名稱。

features.value

支援的功能值。

features.configs

如有需要，此功能的組態設定。

features.config.name

組態設定的名稱。

features.config.value

支援的設定值。

devices

要測試之集區中的裝置陣列。至少需要一個裝置。

`devices.id`

使用者定義的唯一識別符，用於識別要測試的裝置。

`connectivity.protocol`

用來與此裝置通訊的通訊協定。集區中的每個裝置都必須使用相同的通訊協定。

目前，唯一支援的值是實體裝置的 `ssh` 和 `mqtt`，以及 `uart` Docker 容器 `docker` 的 `mqtt` 和 `ssh`。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.port`

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

`connectivity.serialPort`

選用。裝置所連接的序列連接埠。

只有當 `connectivity.protocol` 設為 `uart` 時，才會套用此屬性。

`connectivity.containerId`

要測試之 Docker 容器的容器 ID 或名稱。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.containerUser`

選用。容器內使用者的名稱。預設值是 Dockerfile 中提供的使用者。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

Note

若要檢查測試執行器是否設定不正確的測試裝置連線，您可以從測試協調器內容 `pool.Devices[0].Connectivity.Protocol` 擷取，並將其與 `Choice` 狀態中的預期值進行比較。如果使用不正確的通訊協定，則使用 `LogMessage` 狀態列印訊息並轉換為 `Fail` 狀態。

或者，您可以使用錯誤處理程式碼來報告不正確裝置類型的測試失敗。

(選用) 設定 `userdata.json`

`userdata.json` 檔案包含測試套件所需的任何其他資訊，但未在 `device.json` 檔案中指定。此檔案的格式取決於測試套件中定義的 [userdata_scheme.json 檔案](#)。如果您是測試寫入器，請務必將此資訊提供給將執行您所寫入測試套件的使用者。

(選用) 設定 resource.json

resource.json 檔案包含將用作資源裝置的任何裝置的相關資訊。資源裝置是測試待測裝置的特定功能所需的裝置。例如，若要測試裝置的 Bluetooth 功能，您可以使用資源裝置來測試您的裝置是否能夠成功連線。資源裝置是選用的，您可以視需要要求任意數量的資源裝置。身為測試寫入器，您可以使用 [test.json 檔案](#) 來定義測試所需的資源裝置功能。然後，測試執行器會使用 resource.json 檔案來提供具有所需功能的資源裝置集區。請務必將此資訊提供給將執行您寫入之測試套件的使用者。

測試執行器可以使用位於 `<device-tester-extract-location>/configs/` 資料夾的下列範本 resource.json 檔案提供此資訊。

```
[
  {
    "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>",
      }
    }
  ]
}
```

如下所述，包含值的所有欄位皆為必要：

id

使用者定義的英數字元 ID，可唯一識別裝置的集合，稱為「裝置集區」。屬於同一個集區的裝置必須有相同的硬體。當您執行測試套件，集區中的裝置會用來將工作負載平行化。多個裝置用來執行不同的測試。

features

選用。包含裝置支援功能的陣列。此欄位所需的資訊會在測試套件的 [test.json 檔案中](#) 定義，並決定要執行哪些測試，以及如何執行這些測試。如果測試套件不需要任何功能，則不需要此欄位。

features.name

功能的名稱。

features.version

功能版本。

features.jobSlots

設定 以指示可以同時使用裝置的測試數量。預設值為 1。

devices

要測試之集區中的裝置陣列。至少需要一個裝置。

devices.id

使用者定義的唯一識別符，用於識別要測試的裝置。

connectivity.protocol

用來與此裝置通訊的通訊協定。集區中的每個裝置都必須使用相同的通訊協定。

目前，唯一支援的值是實體裝置的 ssh 和 ，以及 uart Docker 容器 docker 的 和 。

`connectivity.ip`

要測試之裝置的 IP 位址。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.port`

選用。用於 SSH 連線的連接埠號碼。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth`

連線的驗證資訊。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.auth.method`

用來透過指定的連線通訊協定存取裝置的驗證方法。

支援的值如下：

- `pki`
- `password`

`connectivity.auth.credentials`

用於驗證的登入資料。

`connectivity.auth.credentials.password`

用於登入要測試裝置的密碼。

只有當 `connectivity.auth.method` 設為 `password` 時，才會套用此值。

`connectivity.auth.credentials.privKeyPath`

用來登入待測裝置之私有金鑰的完整路徑。

只有當 `connectivity.auth.method` 設為 `pki` 時，才會套用此值。

`connectivity.auth.credentials.user`

登入要測試之裝置的使用者名稱。

`connectivity.serialPort`

選用。裝置所連接的序列連接埠。

只有當 `connectivity.protocol` 設為 `uart` 時，才會套用此屬性。

`connectivity.containerId`

要測試之 Docker 容器的容器 ID 或名稱。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

`connectivity.containerUser`

選用。容器內使用者的名稱。預設值是 Dockerfile 中提供的使用者。

預設值為 22。

只有當 `connectivity.protocol` 設為 `ssh` 時，才會套用此屬性。

(選用) 設定 config.json

`config.json` 檔案包含 IDT 的組態資訊。一般而言，除了提供 IDT AWS 的使用者登入資料，以及選擇性 AWS 的區域之外，測試執行器不需要修改此檔案。如果 AWS 提供具有必要許可的登入資料，會 AWS IoT Device Tester 收集並提交用量指標給 AWS。這是選擇加入功能，用於改善 IDT 功能。如需詳細資訊，請參閱[IDT 用量指標](#)。

測試執行器可以透過下列其中一種方式設定其 AWS 登入資料：

- 憑證檔案

IDT 會使用與 AWS CLI 相同的登入資料檔案。如需詳細資訊，請參閱[組態與登入資料檔案](#)。

登入資料檔案的位置會有所不同，取決於您使用的作業系統：

- macOS, Linux: `~/.aws/credentials`
- Windows : `C:\Users\UserName\.aws\credentials`
- 環境變數

環境變數是由作業系統維護且由系統命令使用的變數。在 SSH 工作階段期間定義的變數在該工作階段關閉後無法使用。IDT 可以使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 環境變數來存放 AWS 登入資料

若要在 Linux、macOS 或 Unix 上設定這些變數，請使用 export：

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要在 Windows 上設定這些變數，請使用 set：

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

若要設定 IDT 的 AWS 登入資料，測試執行器會編輯位於 *<device-tester-extract-location>*/configs/ 資料夾之 config.json 檔案中的 auth 區段。

```
{
  "log": {
    "location": "logs"
  },
  "configFiles": {
    "root": "configs",
    "device": "configs/device.json"
  },
  "testPath": "tests",
  "reportPath": "results",
  "awsRegion": "<region>",
  "auth": {
    "method": "file | environment",
    "credentials": {
      "profile": "<profile-name>"
    }
  }
}
```

如下所述，包含值的所有欄位皆為必要：

Note

此檔案中的所有路徑都是相對於 *<device-tester-extract-location>* 定義。

`log.location`

<*device-tester-extract-location*> 中日誌資料夾的路徑。

`configFiles.root`

包含組態檔案的資料夾路徑。

`configFiles.device`

`device.json` 檔案的路徑。

`testPath`

包含測試套件的資料夾路徑。

`reportPath`

IDT 執行測試套件後，將包含測試結果的資料夾路徑。

`awsRegion`

選用。測試套件將使用 AWS 的區域。如果未設定，則測試套件將使用每個測試套件中指定的預設區域。

`auth.method`

IDT 用來擷取 AWS 登入資料的方法。支援的值 `file` 是從登入資料檔案擷取登入資料，以及使用環境變數 `environment` 擷取登入資料。

`auth.credentials.profile`

從登入資料檔案使用的登入資料設定檔。只有當 `auth.method` 設為 `file` 時，才會套用此屬性。

偵錯並執行自訂測試套件

設定[必要的組態](#)後，IDT 可以執行您的測試套件。完整測試套件的執行時間取決於硬體和測試套件的組成。如需參考，在 Raspberry Pi 3B AWS IoT Greengrass 上完成完整資格測試套件約需 30 分鐘。3B

當您撰寫測試套件時，您可以使用 IDT 在除錯模式下執行測試套件，在執程式碼之前先檢查程式碼，或將其提供給測試執行器。

在除錯模式下執行 IDT

由於測試套件依賴 IDT 與裝置互動、提供內容和接收結果，因此如果沒有任何 IDT 互動，您就無法在 IDE 中對測試套件進行偵錯。若要這樣做，IDT CLI 會提供 `debug-test-suite` 命令，可讓您在偵錯模式下執行 IDT。執行下列命令以檢視 的可用選項 `debug-test-suite`：


```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

當您在偵錯模式下執行 IDT 時，IDT 實際上不會啟動測試套件或執行測試協調器；而是與您的 IDE 互動，以回應在 IDE 中執行的測試套件發出的請求，並將日誌列印到主控台。IDT 不會逾時，並等待結束，直到手動中斷為止。在偵錯模式中，IDT 也不會執行測試協調器，也不會產生任何報告檔案。若要偵錯測試套件，您必須使用 IDE 來提供 IDT 通常從組態 JSON 檔案取得的一些資訊。請務必提供下列資訊：

- 每個測試的環境變數和引數。IDT 不會從 `test.json` 或 讀取此資訊 `suite.json`。
- 選取資源裝置的引數。IDT 不會從 讀取此資訊 `test.json`。

若要偵錯您的測試套件，請完成下列步驟：

1. 建立執行測試套件所需的設定組態檔案。例如，如果您的測試套件需要 `device.json`、`resource.json` 和 `user data.json`，請務必視需要設定所有測試套件。
2. 執行下列命令，將 IDT 置於偵錯模式，然後選取執行測試所需的任何裝置。

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

在您執行此命令之後，IDT 會等待測試套件的請求，然後回應這些請求。IDT 也會產生 IDT 用戶端 SDK 案例程序所需的環境變數。

3. 在您的 IDE 中，使用 `run` 或 `debug` 組態來執行下列動作：
 - a. 設定 IDT 產生環境變數的值。
 - b. 設定您在 `test.json` 和 `suite.json` 檔案中指定的任何環境變數或引數的值。
 - c. 視需要設定中斷點。
4. 在 IDE 中執行測試套件。

您可以視需要多次偵錯並重新執行測試套件。IDT 不會在偵錯模式下逾時。

5. 完成偵錯後，請中斷 IDT 以結束偵錯模式。

執行測試的 IDT CLI 命令

下節說明 IDT CLI 命令：

IDT v4.0.0

`help`

列出所指定命令的相關資訊。

`list-groups`

列出指定測試套件中的群組。

`list-suites`

列出可用的測試套件。

`list-supported-products`

在此情況下，列出 IDT 版本支援的 產品 AWS IoT Greengrass ，以及適用於目前 AWS IoT Greengrass IDT 版本的資格測試套件版本。

`list-test-cases`

列出特定測試群組中的測試案例。支援下列選項：

- `group-id`。要搜尋的測試群組。此選項為必要選項，且必須指定單一群組。

`run-suite`

在裝置集區上執行測試套件。以下是一些常用的選項：

- `suite-id`。要執行的測試套件版本。如果未指定，IDT 會使用 `tests` 資料夾中的最新版本。
- `group-id`。要執行的測試群組，以逗號分隔的清單形式呈現。如果未指定，IDT 會執行測試套件中的所有測試群組。
- `test-id`。要執行的測試案例，以逗號分隔的清單形式呈現。指定時，`group-id` 必須指定單一群組。
- `pool-id`。要測試的裝置集區。如果測試執行器在您的 `device.json` 檔案中定義了多個裝置集區，則必須指定集區。
- `timeout-multiplier`。設定 IDT 修改具有使用者定義乘數之測試的 `test.json` 檔案中指定的測試執行逾時。
- `stop-on-first-failure`。設定 IDT 在第一次失敗時停止執行。此選項應與 `group-id` 搭配使用以偵錯指定的測試群組。
- `userdata`。設定檔案，其中包含執行測試套件所需的使用者資料資訊。只有在測試套件的 `suite.json` 檔案中 `userdataRequired` 設定為 `true` 時，才需要這樣做。

如需 `run-suite` 選項的詳細資訊，請使用下列 `help` 選項：

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

`debug-test-suite`

在除錯模式下執行測試套件。如需詳細資訊，請參閱[在除錯模式下執行 IDT](#)。

檢閱 IDT 測試結果和日誌

本節說明 IDT 產生主控台日誌和測試報告的格式。

主控台訊息格式

AWS IoT Device Tester 使用標準格式，在 主控台啟動測試套件時，將訊息列印到 主控台。以下摘錄顯示 IDT 產生的主控台訊息範例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0  
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多數主控台訊息包含下列欄位：

`time`

已記錄事件的完整 ISO 8601 時間戳記。

`level`

已記錄事件的訊息層級。一般而言，記錄的訊息層級是 `info`、`warn` 或 `error`。如果 IDT 遇到預期事件導致提早結束，則發出 `fatal` 或 `panic` 訊息。

`msg`

記錄的訊息。

`executionId`

目前 IDT 程序的唯一 ID 字串。此 ID 用於區分個別 IDT 執行。

從測試套件產生的主控台訊息提供有關受測裝置以及 IDT 執行的測試套件、測試群組和測試案例的其他資訊。以下摘錄顯示從測試套件產生的主控台訊息範例。

```
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
```

主控台訊息的測試套件特定部分包含下列欄位：

suiteId

目前正在執行的測試套件名稱。

groupId

目前正在執行的測試群組 ID。

testCaseId

目前執行的測試案例 ID。

deviceId

目前測試案例正在使用的測試中裝置的 ID。

若要在 IDT 完成執行測試時將測試摘要列印到主控台，您必須在測試協調器中包含 [Report 狀態](#)。測試摘要包含測試套件的相關資訊、執行的每個群組的測試結果，以及產生的日誌和報告檔案的位置。下列範例顯示測試摘要訊息。

```
===== 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
```

AWS IoT Device Tester 報告結構描述

awsiotdevicetester_report.xml 是已簽章的報告，其中包含下列資訊：

- IDT 版本。
- 測試套件版本。
- 用於簽署報告的報告簽章和金鑰。
- device.json 檔案中指定的裝置 SKU 和裝置集區名稱。
- 已測試的產品版本和裝置功能。
- 測試結果的彙總摘要。此資訊與 *suite-name*_report.xml 檔案中包含的資訊相同。

```
<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>
```

awsiotdevicetester_report.xml 檔案包含 <awsproduct> 標籤，其中包含關於受測產品和經過一系列測試驗證後之產品功能的資訊。

<awsproduct> 標籤中使用的屬性

name

受測產品名稱。

version

受測產品版本。

features

驗證的功能。測試套件required需要標記為 的功能，才能驗證裝置。以下片段顯示此資訊如何出現在 awsiotdevicetester_report.xml 檔案中。

```
<feature name="ssh" value="supported" type="required"></feature>
```

標記為 的功能optional不需要驗證。以下程式碼片段顯示選用功能。

```
<feature name="hsi" value="supported" type="optional"></feature>

<feature name="mqtt" value="not-supported" type="optional"></feature>
```

測試套件報告結構描述

*suite-name*_Result.xml 報告採用 [JUnit XML 格式](#)。您可以將它整合到持續整合和部署平台，例如 [Jenkins](#)、[Bamboo](#) 等。報告包含測試結果的彙總摘要。

```
<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>

```

awsiotdevicetester_report.xml 或 中的報告區段會 *suite-name*_report.xml 列出已執行的測試和結果。

第一個 XML 標籤 <testsuites> 包含測試執行的摘要。例如：

```

<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
  disabled="0">

```

<testsuites> 標籤中使用的屬性

name

測試套件的名稱。

time

執行測試套件所需的時間，以秒為單位。

tests

執行的測試次數。

failures

已執行但未通過的測試次數。

errors

IDT 無法執行的測試次數。

disabled

此屬性未使用，可忽略。

如果測試發生失敗或錯誤，您可以檢閱 `<testsuites>` XML 標籤來識別失敗的測試。`<testsuites>` 標籤內的 `<testsuite>` XML 標籤會顯示測試群組的測試結果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
errors="0" skipped="0">
```

其格式類似於 `<testsuites>` 標籤，但有不使用且可忽略的 `skipped` 屬性。在每個 `<testsuite>` XML 標籤內，測試群組每個執行的測試都有 `<testcase>` 標籤。例如：

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

`<testcase>` 標籤中使用的屬性

name

測試的名稱。

attempts

IDT 執行測試案例的次數。

當測試案例失敗或發生錯誤時，系統就會將 `<failure>` 或 `<error>` 標籤新增至 `<testcase>` 標籤，其中附有相關資訊以利故障診斷。例如：

```
<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>
```


IDT 用量指標

如果您提供具有必要許可的 AWS 登入資料，會 AWS IoT Device Tester 收集並提交用量指標給 AWS。這是選擇加入功能，用於改善 IDT 功能。IDT 會收集下列資訊：

- 用來執行 IDT 的 AWS 帳戶 ID
- 用於執行測試的 IDT AWS CLI 命令
- 執行的測試套件
- *<device-tester-extract-location>* 資料夾的測試套件
- 在裝置集區中設定的裝置數量
- 測試案例名稱和執行時間
- 測試結果資訊，例如測試是否通過、失敗、發生錯誤或被略過
- 測試的產品功能
- IDT 結束行為，例如非預期或提早結束

IDT 傳送的所有資訊也會記錄到 *<device-tester-extract-location>/results/<execution-id>/* 資料夾中 *metrics.log* 的檔案。您可以檢視日誌檔案，以查看在測試執行期間收集的資訊。只有在您選擇收集用量指標時，才會產生此檔案。

若要停用指標集合，您不需要採取其他動作。只要不要存放您的 AWS 登入資料，而且如果您有存放的 AWS 登入資料，請勿設定 *config.json* 檔案來存取這些登入資料。

設定您的 AWS 登入資料

如果您還沒有 AWS 帳戶，則必須[建立一個](#)。如果您已經有 AWS 帳戶，您只需要為您的帳戶[設定必要的許可](#)，以允許 IDT AWS 代表您將用量指標傳送至。

步驟 1：建立 AWS 帳戶

在此步驟中，建立和設定 AWS 帳戶。如果您已擁有 AWS 帳戶，請跳到 [the section called “步驟 2：設定 IDT 的許可”](#)。

如果您沒有 AWS 帳戶，請完成下列步驟來建立一個。

註冊 AWS 帳戶

1. 開啟 <https://portal.aws.amazon.com/billing/signup>。

2. 請遵循線上指示進行。

部分註冊程序需接收來電或簡訊，並在電話鍵盤輸入驗證碼。

當您註冊時 AWS 帳戶，AWS 帳戶根使用者會建立。根使用者有權存取該帳戶中的所有 AWS 服務和資源。作為安全最佳實務，請將管理存取權指派給使用者，並且僅使用根使用者來執行[需要根使用者存取權的任務](#)。

若要建立管理員使用者，請選擇下列其中一個選項。

選擇一種管理管理員的方式	到	根據	您也可以
在 IAM Identity Center (建議)	使用短期憑證存取 AWS。 這與安全性最佳實務一致。有關最佳實務的資訊，請參閱 IAM 使用者指南中的 IAM 安全最佳實務 。	請遵循 AWS IAM Identity Center 使用者指南的 入門 中的說明。	在 AWS Command Line Interface 使用者指南中設定 AWS CLI 以使用來設定 AWS IAM Identity Center 程式設計存取。
在 IAM 中 (不建議使用)	使用長期憑證存取 AWS。	請遵循《IAM 使用者指南》中 建立 IAM 使用者以進行緊急存取 的指示。	請依照《IAM 使用者指南》中的 管理 IAM 使用者的存取金鑰 設定以程式設計方式存取。

步驟 2：設定 IDT 的許可

在此步驟中，設定 IDT 用來執行測試和收集 IDT 用量資料的許可。您可以使用 AWS 管理主控台 或 AWS Command Line Interface (AWS CLI) 來建立 IAM 政策和 IDT 的使用者，然後將政策連接到使用者。

- [設定 IDT \(主控台\) 的許可](#)
- [步驟 2：設定 IDT \(AWS CLI\) 的許可](#)

設定 IDT (主控台) 的許可

請依照下列步驟使用主控台來設定 IDT for AWS IoT Greengrass 的許可。

1. 登入 [IAM 主控台](#)。
2. 建立客戶受管政策，該政策授與建立具有特定許可之角色的許可。
 - a. 在導覽窗格中，選擇政策，然後選擇建立政策。
 - b. 在 JSON 標籤上，用以下政策取代預留位置內容。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. 選擇檢閱政策。
 - d. 對於名稱，輸入 **IDTUsageMetricsIAMPermissions**。在 Summary (摘要) 下，檢閱您的政策所授與的許可。
 - e. 選擇建立政策。
3. 建立 IAM 使用者並將許可連接到使用者。
 - a. 建立 IAM 使用者。請遵循《[IAM 使用者指南](#)》中建立 IAM 使用者 (主控台) 的步驟 1 到 5。如果您已建立 IAM 使用者，請跳至下一個步驟。
 - b. 將許可連接至您的 IAM 使用者：
 - i. 在 Set permissions (設定許可) 頁面上，選擇 Attach existing policies directly (直接連接現有的政策)。
 - ii. 搜尋您在上一個步驟中建立的 IDTUsageMetricsIAMPermissions 政策。選取核取方塊。
 - c. 選擇下一步：標籤。

- d. 選擇 Next: Review (下一步：檢閱) 以檢視選擇的摘要。
- e. 選擇 Create user (建立使用者)。
- f. 若要檢視使用者的存取金鑰 (存取金鑰 ID 和私密存取金鑰)，請選擇密碼和存取金鑰旁的 Show (顯示)。若要儲存存取金鑰，請選擇 Download.csv，並將檔案儲存到安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

步驟 2：設定 IDT (AWS CLI) 的許可

請依照下列步驟使用 AWS CLI 來設定 IDT for 的許可 AWS IoT Greengrass。

1. 如果尚未安裝，請在電腦上安裝和設定 AWS CLI。請遵循AWS Command Line Interface 《使用者指南》中的[安裝 AWS CLI](#) 中的步驟。

Note

AWS CLI 是開放原始碼工具，可用來與命令列 Shell 中的 AWS 服務互動。

2. 建立下列客戶受管政策，授予管理 IDT 和 AWS IoT Greengrass 角色的許可。

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

此步驟包含 Windows 命令提示字元範例，因為它使用的 JSON 語法與 Linux、macOS 或 Unix 終端機命令不同。

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}'
```

3. 建立 IAM 使用者並連接 IDT for 所需的許可 AWS IoT Greengrass。

a. 建立 IAM 使用者。

```
aws iam create-user --user-name user-name
```

b. 將您建立IDTUsageMetricsIAMPermissions的政策連接至您的 IAM 使用者。將 *user-name* 取代為您的 IAM 使用者名稱，並將 命令中的 *<account-id>* 取代為您的 ID AWS 帳戶。

```
aws iam attach-user-policy --user-name user-name --policy-arn  
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. 為使用者建立私密存取金鑰。

```
aws iam create-access-key --user-name user-name
```

將輸出儲存在安全的位置。您稍後會使用此資訊來設定您的 AWS 登入資料檔案。

提供 AWS 登入資料給 IDT

若要允許 IDT 存取您的 AWS 登入資料並提交指標 AWS，請執行下列動作：

1. 將 IAM 使用者的 AWS 登入資料儲存為環境變數或儲存在登入資料檔案中：
 - a. 若要使用環境變數，請執行下列命令。

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. 若要使用登入資料檔案，請將下列資訊新增至~/.aws/credentials檔案。

```
[profile-name]  
aws_access_key_id=access-key  
aws_secret_access_key=secret-access-key
```

2. 設定 config.json 檔案的 auth 區段。如需詳細資訊，請參閱 [\(選用\) 設定 config.json](#)。

對 IDT for AWS IoT Greengrass V2 進行故障診斷

IDT for AWS IoT Greengrass V2 會根據錯誤類型將錯誤寫入各種位置。IDT 會將錯誤寫入主控台、日誌檔案和測試報告。

在哪裡尋找錯誤

當測試執行時，主控台會顯示高階錯誤，而失敗測試的摘要會在所有測試完成時顯示。`awsiotdevicetester_report.xml` 包含導致測試失敗之所有錯誤的摘要。IDT 會將每個測試執行的日誌檔案存放在具有測試執行 UUID 的目錄中，並在測試執行期間顯示在主控台上。

IDT 測試日誌目錄為 `<device-tester-extract-location>/results/<execution-id>/logs/`。此目錄包含資料表中顯示的下列檔案。這適用於除錯。

檔案	Description
<code>test_manager.log</code>	測試執行時寫入主控台的日誌。此檔案結尾的結果摘要包含測試失敗的清單。 這個檔案中的警告和錯誤日誌提供有關失敗的一些資訊。
<code>test-group-id /test-case-id /test-name .log</code>	測試群組中特定測試的詳細日誌。對於部署 Greengrass 元件的測試，測試案例日誌檔案稱為 <code>greengrass-test-run.log</code> 。
<code>test-group-id /test-case-id /greengrass.log</code>	AWS IoT Greengrass Core 軟體的詳細日誌。IDT 會在裝置上執行安裝 AWS IoT Greengrass Core 軟體的測試時，從待測裝置複製此檔案。如需此日誌檔案中訊息的詳細資訊，請參閱 故障診斷 AWS IoT Greengrass V2 。
<code>test-group-id /test-case-id/component-name .log</code>	測試執行期間部署之 Greengrass 元件的詳細日誌。IDT 會在執行部署特定元件的測試時，從待測裝置複製元件日誌檔案。每個元件日誌檔的名稱對應於已部署元件的名稱。如需此日誌檔案中訊息的詳細資訊，請參閱 故障診斷 AWS IoT Greengrass V2 。

解決 IDT for AWS IoT Greengrass V2 錯誤

執行 IDT for 之前 AWS IoT Greengrass，請先取得正確的組態檔案。如果您收到剖析和組態錯誤，您的第一步是找出並使用適合您環境的組態範本。

如果您仍然有問題，請參閱下列除錯程序。

主題

- [別名解析錯誤](#)
- [衝突錯誤](#)
- [無法開始測試錯誤](#)
- [Docker 資格映像存在錯誤](#)
- [無法讀取登入資料](#)
- [PreInstalled Greengrass 的 Guice 錯誤](#)
- [無效的簽章例外狀況](#)
- [機器學習資格錯誤](#)
- [開放測試架構 \(OTF\) 失敗的部署](#)
- [剖析錯誤](#)
- [拒絕許可錯誤](#)
- [資格報告產生錯誤](#)
- [遺漏必要參數錯誤](#)
- [macOS 上的安全例外狀況](#)
- [SSH 連線錯誤](#)
- [串流管理員資格錯誤](#)
- [逾時錯誤](#)
- [版本檢查錯誤](#)

別名解析錯誤

當您執行自訂測試套件時，您可能會在 主控台和 中看到下列錯誤test_manager.log。

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```


當 IDT 測試協調器中設定的別名未正確解析，或已解析的值不存在於組態檔案中時，就會發生此錯誤。若要解決此錯誤，請確定您的 `device.json` 和 `userdata.json` 包含測試套件所需的正確資訊。如需 AWS IoT Greengrass 資格所需組態的資訊，請參閱 [設定 IDT AWS IoT Greengrass 設定以執行資格套件](#)。

衝突錯誤

當您同時在多個裝置上執行 AWS IoT Greengrass 資格套件時，您可能會看到下列錯誤。

```
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]" }
```

AWS IoT Greengrass 資格套件尚未支援並行測試執行。為每個裝置依序執行資格套件。

無法開始測試錯誤

您可能會遇到錯誤，指出測試嘗試啟動時發生的失敗。有數種可能的原因，因此，請執行下列動作：

- 請確定執行命令中的集區名稱確實存在。IDT 會直接從您的 `device.json` 檔案參考集區名稱。
- 確保您集區中的裝置都有正確的組態參數。

Docker 資格映像存在錯誤

Docker 應用程式管理員資格測試使用 Amazon ECR 中的 `amazon/amazon-ec2-metadata-mock` 容器映像來限定測試中的裝置。

如果影像已存在於待測裝置的 Docker 容器中，您可能會收到下列錯誤。

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

如果您先前已下載此映像並在裝置上執行 `amazon/amazon-ec2-metadata-mock` 容器，請務必在執行資格測試之前，從受測裝置移除此映像。

無法讀取登入資料

測試 Windows 裝置時，如果您用來連線至待測裝置的使用者未在該裝置的登入資料管理員中設定，則可能會在 `greengrass.log` 檔案中遇到 `Failed to read credential` 錯誤。

若要解決此錯誤，請在待測裝置的登入資料管理員中設定 IDT 使用者的使用者名稱和密碼。

如需詳細資訊，請參閱[設定 Windows 裝置的使用者登入資料](#)。

PreInstalled Greengrass 的 Guice 錯誤

使用 PreInstalled Greengrass 執行 IDT 時，如果您遇到 Guice 或 的錯誤 `ErrorInCustomProvider`，請檢查檔案是否已 `userdata.json` 將 `InstalledDirRootOnDevice` 設定為 Greengrass 安裝資料夾。IDT 會在 `effectiveConfig.yaml` 下檢查 檔案 `<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`。

如需詳細資訊，請參閱[設定 Windows 裝置的使用者登入資料](#)。

無效的簽章例外狀況

當您執行 Lambda 資格測試時，如果您的 IDT 主機機器遇到網路存取問題，您可能會遇到 `invalidsignatureexception` 錯誤。重設路由器並再次執行測試。

機器學習資格錯誤

當您執行機器學習 (ML) 資格測試時，如果裝置不符合部署 AWS 所提供 ML 元件 [的需求](#)，您可能會遇到資格失敗。若要疑難排解 ML 資格錯誤，請執行下列動作：

- 在元件日誌中尋找測試執行期間所部署元件的錯誤詳細資訊。元件日誌位於 `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` 目錄中。
- 將 `-Dgg.persist=installed.software` 引數新增至失敗測試案例的 `test.json` 檔案。 `test.json` 檔案位於 `<device-tester-extract-location>/tests/GGV2Q_version directory`。

開放測試架構 (OTF) 失敗的部署

如果 OTF 測試無法完成部署，可能原因可能是為 `TempResourcesDirOnDevice` 和 的父資料夾設定許可 `InstallationDirRootOnDevice`。若要正確設定此資料夾的許可，請執行下列命令。`folder-name` 將取代為父資料夾的名稱。

```
sudo chmod 755 folder-name
```

剖析錯誤

JSON 組態中的 Typos 可能會導致剖析錯誤。在大部分的情況下，問題是出在 JSON 檔案中省略了括弧、逗號或引號。IDT 會執行 JSON 驗證並列印除錯資訊。該工具會印出發生錯誤的行、行號和語

法錯誤的欄號。此資訊應足以協助您修正錯誤，但如果您仍然找不到錯誤，您可以在 IDE、Atom 或 Sublime 等文字編輯器，或透過 JSONLint 等線上工具手動執行驗證。

拒絕許可錯誤

IDT 會在待測裝置的各種目錄和檔案上執行操作。這些操作中有些需要根存取。IDT 必須能夠在不輸入密碼的情況下使用 `sudo` 執行命令，才能自動化這些操作。

依照以下步驟，在不輸入密碼的情況下允許 `sudo` 存取。

Note

`user` 和 `username` 是指 IDT 存取待測裝置時所使用的 SSH 使用者。

1. 使用 `sudo usermod -aG sudo <ssh-username>` 將您的 SSH 使用者新增至 `sudo` 群組。
2. 登出後再登入，以使變更生效。
3. 開啟 `/etc/sudoers` 檔案，然後在檔案結尾處新增以下一行：`<ssh-username> ALL=(ALL) NOPASSWD: ALL`

Note

最佳實務為建議您在編輯 `/etc/sudoers` 時使用 `sudo visudo`。

資格報告產生錯誤

IDT 支援四個 *major.minor* 最新版本的 AWS IoT Greengrass V2 資格套件 (GGV2Q)，以產生資格報告，您可以提交至 AWS Partner Network 以將裝置包含在 AWS Partner Device Catalog 中。舊版的資格套件不會產生資格報告。

如果您對支援政策有任何疑問，請聯絡 [AWS 支援](#)。

遺漏必要參數錯誤

當 IDT 新增新功能時，可能會將變更引入組態檔案。使用舊的組態檔案可能會破壞組態。如果發生這種情況，`/results/<execution-id>/logs` 下的 `<test_case_id>.log` 檔案會明確列出所有遺漏的參數。IDT 也會驗證您的 JSON 組態檔案結構描述，以確認您使用的是最新的支援版本。

macOS 上的安全例外狀況

當您在 macOS 主機電腦上執行 IDT 時，它會封鎖 IDT 執行。若要執行 IDT，請將安全例外授予 IDT 執行期功能一部分的可執行檔。當您在主機電腦上看到警告訊息顯示時，請針對每個適用的可執行檔執行下列動作：

將安全性例外狀況授予 IDT 可執行檔

1. 在 macOS 電腦上的 Apple 功能表上，開啟系統偏好設定。
2. 選擇安全性與隱私權，然後在一般索引標籤上，選擇鎖定圖示以變更安全性設定。
3. 如果遭到封鎖 `devicetester_mac_x86-64`，請尋找訊息 "`devicetester_mac_x86-64` was blocked from use because it is not from an identified developer."，然後選擇允許任何方式。
4. 繼續 IDT 測試，直到您完成所有涉及的可執行檔為止。

SSH 連線錯誤

當 IDT 無法連線至受測裝置時，它會在 `中記錄連線失敗/results/<execution-id>/logs/<test-case-id>.log`。SSH 訊息會顯示在此日誌檔案的頂端，因為連線至待測裝置是 IDT 執行的第一個操作之一。

大多數 Windows 組態使用 PuTTY 終端機應用程式連線到 Linux 主機。此應用程式要求您將標準 PEM 私有金鑰檔案轉換為名為 PPK 的專屬 Windows 格式。如果您在 `device.json` 檔案中設定 SSH，請使用 PEM 檔案。如果您使用 PPK 檔案，IDT 無法與 AWS IoT Greengrass 裝置建立 SSH 連線，也無法執行測試。

從 IDT v4.4.0 開始，如果您尚未在待測裝置上啟用 SFTP，則您可能會在日誌檔案中看到下列錯誤。

```
SSH connection failed with EOF
```

若要解決此錯誤，請在裝置上啟用 SFTP。

串流管理員資格錯誤

當您執行串流管理員資格測試時，您可能會在 `com.aws.StreamManagerExport.log` 檔案中看到下列錯誤。

```
Failed to upload data to S3
```

當串流管理員使用您裝置上的 `~/root/.aws/credentials` 檔案中的 AWS 登入資料，而不是使用 IDT 匯出到待測裝置的環境登入資料時，可能會發生此錯誤。若要避免此問題，請刪除裝置上的 `credentials` 檔案，然後重新執行資格測試。

逾時錯誤

您可以指定套用至每個測試逾時預設值的逾時乘數，以增加每個測試的逾時。此旗標設定的任何值必須大於或等於 1.0。

若要使用逾時乘數，請在執行測試時使用旗標 `--timeout-multiplier`。例如：

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

如需詳細資訊，請執行 `run-suite --help`。

當 IDT 測試案例因為組態問題而無法完成時，會發生一些逾時錯誤。您無法透過增加逾時乘數來解決這些錯誤。使用測試執行的日誌來疑難排解基礎組態問題。

- 如果 MQTT 或 Lambda 元件日誌包含 `Access denied` 錯誤，您的 Greengrass 安裝資料夾可能沒有正確的檔案許可。針對您在 `userdata.json` 檔案中定義的安裝路徑中的每個資料夾執行下列命令。

```
sudo chmod 755 folder-name
```

- 如果 Greengrass 日誌指出 Greengrass CLI 部署未完成，請執行下列動作：
 - 確認 `bash` 已安裝在待測裝置上。
 - 如果您的 `userdata.json` 檔案包含 `GreengrassCliVersion` 組態參數，請將其移除。此參數在 IDT v4.1.0 和更新版本中已棄用。如需詳細資訊，請參閱[設定 userdata.json](#)。
- 如果 Lambda 部署測試失敗，並顯示錯誤訊息「驗證 Lambda 發佈：逾時」，而且您在測試日誌檔案中收到錯誤，指出 `(idt-gg2-lambda-function-idt-<resource-id>.log)Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.`，請執行下列動作：
 - 驗證 `userdata.json` `InstallationDirRootOnDevice` 檔案中用於哪些資料夾。
 - 請確定您的裝置已設定正確的使用者許可。如需詳細資訊，請參閱[在裝置上設定使用者許可](#)。

版本檢查錯誤

當 IDT AWS 使用者的使用者登入資料沒有必要的 IAM 許可時，IDT 會發出下列錯誤。

```
Failed to check version compatibility
```

沒有必要 IAM 許可 AWS 的使用者。

適用於 AWS IoT Device Tester 的支援政策 AWS IoT Greengrass

AWS IoT Device Tester for AWS IoT Greengrass 是測試自動化工具，用於驗證 AWS IoT Greengrass 您的裝置，並符合[AWS Partner 包含在 Device Catalog 中的資格](#)。我們建議您使用最新版本的 AWS IoT Greengrass 和 AWS IoT Device Tester 來測試或驗證您的裝置。

每個支援的版本至少 AWS IoT Device Tester 有一個版本可用 AWS IoT Greengrass。如需支援的版本 AWS IoT Greengrass，請參閱 [Greengrass 核版本](#)。如需支援的版本 AWS IoT Device Tester，請參閱 [for AWS IoT Greengrass V2 AWS IoT Device Tester 支援的版本](#)。

您也可以使用任何支援的 AWS IoT Greengrass 和版本 AWS IoT Device Tester 來測試或驗證您的裝置。雖然您可以繼續使用不支援的版本 AWS IoT Device Tester，但這些版本不會收到錯誤修正或更新。如果您對支援政策有任何疑問，請聯絡 [AWS 支援](#)。

Greengrass 型 IoT 解決方案

Eurotech 的 Everyware GreenEdge 正處於的預覽版本中，AWS IoT Greengrass 可能會有所變更。不支援此解決方案 AWS。如果此裝置有任何問題，您必須聯絡 Eurotech。

AWS IoT Greengrass 提供合作夥伴的解決方案，以最佳化您的 Greengrass 安裝體驗。以下是 AWS 與 Eurotech 合作提供的解決方案。此解決方案隨附 AWS IoT Greengrass Core 節點執行期和預先安裝的其他功能。

Eurotech

AWS 已與 Eurotech 合作，為正在尋找預先安裝 AWS IoT Greengrass Core 軟體之裝置的客戶提供 IoT 解決方案。Eurotech 的 Everyware GreenEdge 是一種 IoT 邊緣軟體，由預先設定並預先符合資格 AWS。此解決方案結合了 Greengrass 和 Eurotech Everyware Software Framework (ESF) 的功能，透過通訊協定轉接器為客戶提供廣泛的南向連線，例如：Modbus、OPC-UA Client/Server、S7、TwinCAT、J1939, DNP3 Master/Outstation 等。使用此解決方案，您也可以將資料傳送至，AWS 雲端並連線至所有北向 AWS 服務（例如 AWS IoT Core、AWS IoT SiteWise、AWS IoT Analytics、Amazon S3 和 Amazon Kinesis Video Streams）。結合 Eurotech 的裝置管理解決方案 Everyware Cloud，此解決方案推出全新的 Zero-Touch 佈建服務，可簡化裝置加入和大量部署。

如需 Eurotech 的詳細資訊，請參閱 [Eurotech](#)。

故障診斷 AWS IoT Greengrass V2

使用本節中的疑難排解資訊和解決方案，以協助解決的問題 AWS IoT Greengrass Version 2。

主題

- [檢視 AWS IoT Greengrass 核心軟體和元件日誌](#)
- [AWS IoT Greengrass 核心軟體問題](#)
- [AWS IoT Greengrass 雲端問題](#)
- [核心裝置部署問題](#)
- [核心裝置元件問題](#)
- [核心裝置 Lambda 函數元件問題](#)
- [元件版本已停止](#)
- [Greengrass 命令列界面問題](#)
- [AWS Command Line Interface 問題](#)
- [詳細部署錯誤代碼](#)
- [詳細的元件狀態碼](#)

檢視 AWS IoT Greengrass 核心軟體和元件日誌

Core AWS IoT Greengrass 軟體會將日誌寫入本機檔案系統，供您用來檢視核心裝置的即時資訊。您也可以設定核心裝置將日誌寫入 CloudWatch Logs，以便遠端疑難排解核心裝置。這些日誌可協助您識別元件、部署和核心裝置的問題。如需詳細資訊，請參閱[監控 AWS IoT Greengrass 日誌](#)。

AWS IoT Greengrass 核心軟體問題

針對 AWS IoT Greengrass Core 軟體問題進行故障診斷。

主題

- [ListDeployments API 中的 ThrottlingException](#)
- [無法設定核心裝置](#)
- [無法啟動 AWS IoT Greengrass Core 軟體做為系統服務](#)

- [無法將 核設定為系統服務](#)
- [無法連線至 AWS IoT Core](#)
- [記憶體不足錯誤](#)
- [無法安裝 Greengrass CLI](#)
- [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](#)
- [無法設定 Windows 服務](#)
- [com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager](#)
- [com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot 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](#)

ListDeployments API 中的 ThrottlingException

ThrottlingException 從 ListDeployments API：當您的帳戶中有大量部署時，您可能會看到此訊息。

若要解決此問題，請執行下列其中一項操作：

- 如果您使用 SDK，請指定 MaxResult 參數。例如，對於具有小值（例如 5）的 [JavaSDK](#)。
- 您可以使用 [AWS Service Quotas](#) 來請求提高 API 速率限制 DescribeJob。您可以前往服務配額主控台，選取的配額，AWS IoT 且限制名稱為 DescribeJob 限流限制。您可以將其從 10 增加到 50。

無法設定核心裝置

如果 AWS IoT Greengrass 核心軟體安裝程式失敗，而且您無法設定核心裝置，您可能需要解除安裝軟體，然後再試一次。如需詳細資訊，請參閱[解除安裝 AWS IoT Greengrass Core 軟體](#)。

無法啟動 AWS IoT Greengrass Core 軟體做為系統服務

如果 AWS IoT Greengrass 核心軟體無法啟動，[請檢查系統服務日誌](#)以識別問題。其中一個常見問題是，PATH 環境變數 (Linux) 或 PATH 系統變數 (Windows) 上無法使用 Java。

無法將 核設定為系統服務

當 AWS IoT Greengrass Core 軟體安裝程式無法設定為 AWS IoT Greengrass 系統服務時，您可能會看到此錯誤。在 Linux 裝置上，如果核心裝置沒有[系統化](#)初始化系統，通常會發生此錯誤。即使安裝程式無法設定系統服務，也能成功設定 AWS IoT Greengrass Core 軟體。

執行以下任意一項：

- 設定並執行 AWS IoT Greengrass Core 軟體做為系統服務。您必須將軟體設定為系統服務，以使用的所有功能 AWS IoT Greengrass。您可以安裝[系統化](#)或使用不同的初始化系統。如需詳細資訊，請參閱[將 Greengrass 核設定為系統服務](#)。
- 在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體。您可以使用安裝程式在 Greengrass 根資料夾中設定的載入器指令碼來執行軟體。如需詳細資訊，請參閱[在沒有系統服務的情況下執行 AWS IoT Greengrass Core 軟體](#)。

無法連線至 AWS IoT Core

例如，當 AWS IoT Greengrass 核心軟體無法連線至 AWS IoT Core 以擷取部署任務時，您可能會看到此錯誤。請執行下列操作：

- 檢查您的核心裝置是否可以連線至網際網路和 AWS IoT Core。如需裝置所連線 AWS IoT Core 端點的詳細資訊，請參閱 [設定 AWS IoT Greengrass 核心軟體](#)。
- 檢查核心裝置的 AWS IoT 物件是否使用允許 `iot:Connect`、`iot:Receive`、`iot:Publish` 和 `iot:Subscribe` 許可的憑證。
- 如果您的核心裝置使用[網路代理](#)，請檢查您的核心裝置是否具有[裝置角色](#)，以及其角色是否允許 `iot:Connect`、`iot:Receive`、`iot:Publish` 和 `iot:Subscribe` 許可。

記憶體不足錯誤

如果您的裝置沒有足夠的記憶體在 Java 堆積中配置物件，通常會發生此錯誤。在記憶體有限的裝置上，您可能需要指定堆積大小上限來控制記憶體配置。如需詳細資訊，請參閱[使用 JVM 選項控制記憶體配置](#)。

無法安裝 Greengrass CLI

當您在適用於 AWS IoT Greengrass Core 的安裝命令中使用 `--deploy-dev-tools` 引數時，您可能會看到下列主控台訊息。

```
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
```

未安裝 Greengrass CLI 元件時會發生這種情況，因為您的核心裝置是具有現有部署之物件群組的成員。如果您看到此訊息，您可以將 Greengrass CLI 元件 (`aws.greengrass.Cli`) 手動部署到裝置，以安裝 Greengrass CLI。如需詳細資訊，請參閱[安裝 Greengrass CLI](#)。

User root is not allowed to execute

當執行 AWS IoT Greengrass 核心軟體的使用者（通常是 `root`）沒有 `sudo` 使用任何使用者和任何群組執行的許可時，您可能會看到此錯誤。對於預設 `ggc_user` 系統使用者，此錯誤如下所示：

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

檢查您的 `/etc/sudoers` 檔案是否授予使用者執行 `sudo` 做為其他群組的許可。中使用者的許可/`etc/sudoers`應如下所示。

```
root    ALL=(ALL:ALL) ALL
```

`com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with`

當核心裝置嘗試執行元件時，您可能會看到此錯誤，Greengrass 核不會指定用於執行元件的預設系統使用者。

若要修正此問題，請設定 Greengrass 核以指定執行元件的預設系統使用者。如需詳細資訊，請參閱[設定執行元件的使用者](#)及[設定預設元件使用者](#)。

`Failed to map segment from shared object: operation not permitted`

當 AWS IoT Greengrass Core 軟體因為 `/tmp` 資料夾掛載有 `noexec` 許可而無法啟動時，您可能會看到此錯誤。根據預設，[AWS 通用執行期 \(CRT\) 程式庫](#)會使用 `/tmp` 資料夾。

執行以下任意一項：

- 執行下列命令來重新掛載具有 `exec` 許可的 `/tmp` 資料夾，然後再試一次。

```
sudo mount -o remount,exec /tmp
```

- 如果您執行 Greengrass 核 v2.5.0 或更新版本，您可以設定 JVM 選項來變更 AWS CRT 程式庫使用的資料夾。您可以在部署或安裝 AWS IoT Greengrass Core 軟體時，於 Greengrass 核元件組態中指定 `jvmOptions` 參數。將 `/path/to/use` 取代為 AWS CRT 程式庫可以使用的資料夾路徑。

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

無法設定 Windows 服務

如果您在 Microsoft Windows 2016 裝置上安裝 AWS IoT Greengrass 核心軟體，您可能會看到此錯誤。Windows 2016 不支援 AWS IoT Greengrass Core 軟體，如需支援的作業系統清單，請參閱[支援平台](#)。

如果您必須使用 Windows 2016，您可以執行下列動作：

1. 解壓縮下載的核心 AWS IoT Greengrass 安裝封存
2. 在 Greengrass 目錄中開啟 bin/greengrass.xml.template 檔案。
3. 將<autoRefresh>標籤新增至 檔案的結尾，就在</service>標籤前面。

```
</log>
<autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

當您在沒有根憑證授權機構 (CA) 檔案安裝 AWS IoT Greengrass Core 軟體時，可能會看到此錯誤。

```
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
```

檢查您是否使用您提供給安裝程式的組態檔案中的 rootCaPath 參數指定有效的根 CA 檔案。如需詳細資訊，請參閱[安裝 AWS IoT Greengrass Core 軟體](#)。

com.aws.greengrass.deployment.lotJobsHelper: No connection available during subscribing to lot Jobs descriptions topic. Will retry in sometime

當核心裝置無法連線至 AWS IoT Core 以訂閱部署任務通知時，您可能會看到此警告訊息。請執行下列操作：

- 檢查核心裝置是否已連線至網際網路，並且可以連線到您設定 AWS IoT 的資料端點。如需核心裝置使用之端點的詳細資訊，請參閱[允許透過代理或防火牆的裝置流量](#)。
- 檢查 Greengrass 日誌是否有其他顯示其他根本原因的錯誤。

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

當您[使用自動佈建安裝 AWS IoT Greengrass 核心軟體時](#)，您可能會看到此錯誤，安裝程式會使用無效的 AWS 工作階段字串。請執行下列操作：

- 如果您使用臨時安全登入資料，請检查工作階段字串是否正確，以及您正在複製和貼上完整的工作階段字串。
- 如果您使用長期安全登入資料，請檢查裝置在您先前使用臨時登入資料時是否沒有工作階段字串。請執行下列操作：

1. 執行下列命令來取消設定工作階段字串環境變數。

Linux or Unix

```
unset AWS_SESSION_TOKEN
```

Windows Command Prompt (CMD)

```
set AWS_SESSION_TOKEN=
```

PowerShell

```
Remove-Item Env:\AWS_SESSION_TOKEN
```

2. 檢查 AWS 登入資料檔案 是否~/.aws/credentials 包含工作階段字串 aws_session_token。若是如此，請從 檔案移除該行。

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwmqKwRc0IfRrh3c/LTo6UDdyJw00vEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJ0gQs8IZZaIv2BXIa2R40lgk
```

您也可以安裝 AWS IoT Greengrass Core 軟體，而無需提供 AWS 登入資料。如需詳細資訊，請參閱 [使用手動資源佈建安裝 AWS IoT Greengrass 核心軟體](#) 或 [使用 AWS IoT 機群佈建安裝 AWS IoT Greengrass 核心軟體](#)。

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

當您[使用自動佈建安裝 AWS IoT Greengrass 核心軟體時](#)，您可能會看到此錯誤，安裝程式會使用沒有必要許可的 AWS 登入資料。如需所需許可的詳細資訊，請參閱[安裝程式佈建資源的最低 IAM 政策](#)。

檢查憑證 IAM 身分的許可，並授予 IAM 身分任何缺少的必要許可。

Error:

com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest:
Could not execute cloud shadow get request

當您使用[影子管理員元件](#)來[同步裝置影 AWS IoT Core](#)子時，您可能會看到此錯誤。HTTP 403 狀態碼表示發生此錯誤，因為核心裝置的 AWS IoT 政策未授予呼叫的許可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)
```

若要與 同步本機影子 AWS IoT Core，核心裝置的 AWS IoT 政策必須授予下列許可：

- iot:GetThingShadow
- iot:UpdateThingShadow
- iot:DeleteThingShadow

檢查核心裝置的 AWS IoT 政策，並新增任何缺少的必要許可。如需詳細資訊，請參閱下列內容：

- 《AWS IoT 開發人員指南》中的[AWS IoT Core 政策動作](#)
- [更新核心裝置的 AWS IoT 政策](#)

Operation aws.greengrass#<operation> is not supported by Greengrass

當您在自訂 Greengrass 元件中使用[程序間通訊 \(IPC\) 操作](#)，且核心裝置上未安裝必要 AWS 提供的元件時，您可能會看到此錯誤。

若要修正此問題，請在[元件配方中新增必要的元件做為相依性](#)，因此 AWS IoT Greengrass Core 軟體會在您部署元件時安裝必要的元件。

- [擷取秘密值](#) – aws.greengrass.SecretManager
- [與本機陰影互動](#) – aws.greengrass.ShadowManager
- [管理本機部署和元件](#) – aws.greengrass.Cli v2.6.0 或更新版本
- [驗證和授權用戶端裝置](#) – v2aws.greengrass.clientdevices.Auth.2.0 或更新版本

java.io.FileNotFoundException: <stream-manager-store-root-dir>/
stream_manager_metadata_store (Permission denied)

當您設定串流管理員使用不存在或具有正確許可的根資料夾時，您可能會在[串流管理員](#)日誌檔 (aws.greengrass.StreamManager.log) 中看到此錯誤。如需如何設定此資料夾的詳細資訊，請參閱[串流管理員組態](#)。

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService:
Private key or certificate with label <label> does not exist

當 [PKCS#11 提供者元件](#) 找不到或載入您在設定 AWS IoT Greengrass 核心軟體使用[硬體安全模組 \(HSM\)](#) 時指定的私有金鑰或憑證時，就會發生此錯誤。請執行下列操作：

- 檢查私有金鑰和憑證是否使用您設定 AWS IoT Greengrass 核心軟體使用的插槽、使用者 PIN 和物件標籤存放在 HSM 中。
- 檢查私有金鑰和憑證是否在 HSM 中使用相同的物件標籤。
- 如果您的 HSM 支援物件 IDs，請檢查私有金鑰和憑證是否在 HSM 中使用相同的物件 ID。

檢查您的 HSM 文件，了解如何查詢 HSM 中安全字符的詳細資訊。如果您需要變更安全字符的槽、物件標籤或物件 ID，請檢查 HSM 的文件以了解如何執行此操作。

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

當您使用[秘密管理員元件](#)來部署 AWS Secrets Manager 秘密時，可能會發生此錯誤。如果核心裝置的[字符交換 IAM 角色](#)未授予取得秘密的許可，則部署會失敗，且 Greengrass 日誌會包含此錯誤。

授權核心裝置下載秘密

1. 將 `secretsmanager:GetSecretValue` 許可新增至核心裝置的字符交換角色。下列範例政策陳述式會授予取得秘密值的許可。

```
{
  "Effect": "Allow",
  "Action": [
    "secretsmanager:GetSecretValue"
  ],
  "Resource": [
    "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-abcdef"
  ]
}
```

如需詳細資訊，請參閱[授權核心裝置與 AWS 服務互動](#)。

2. 將部署重新套用至核心裝置。執行以下任意一項：

 - 修改部署而不進行任何變更。核心裝置在收到修訂後的部署時，會嘗試再次下載秘密。如需詳細資訊，請參閱[修訂部署](#)。
 - 重新啟動 AWS IoT Greengrass Core 軟體以重試部署。如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)

如果秘密管理員成功下載秘密，則部署會成功。

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed

當您使用[秘密管理員元件](#)來部署由金鑰加密的 AWS Secrets Manager AWS Key Management Service 秘密時，可能會發生此錯誤。如果核心裝置的[字符交換 IAM 角色](#)未授予解密秘密的許可，則部署會失敗，且 Greengrass 日誌包含此錯誤。

若要修正此問題，請將 kms:Decrypt 許可新增至核心裝置的字符交換角色。如需詳細資訊，請參閱下列內容：

- AWS Secrets Manager 《使用者指南》中的[秘密加密和解密](#)
- [授權核心裝置與 AWS 服務互動](#)

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

當您嘗試安裝具有[硬體安全性](#)的 AWS IoT Greengrass Core 軟體，並使用不支援硬體安全性整合的舊版 Greengrass 核心版本時，您可能會看到此錯誤。若要使用硬體安全整合，您必須使用 Greengrass nucleus v2.5.3 或更新版本。

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

當您在執行 AWS IoT Greengrass Core 做為系統服務時使用 TPM2 程式庫時，您可能會看到此錯誤。

此錯誤表示您需要新增環境變數，以提供 AWS IoT Greengrass 核心系統化服務檔案中 PKCS#11 存放區的位置。

如需詳細資訊，請參閱[PKCS#11 供應商](#)元件文件的要求一節。

Greengrass core device stuck on nucleus v2.12.3

如果您的 Greengrass 核心裝置不會從 2.12.3 版修改部署，您可能需要下載並以 Greengrass 2.12.2 版取代 Greengrass.jar 檔案。請執行下列操作：

1. 在您的 Greengrass 核心裝置上，執行下列命令來停止 Greengrass Core 軟體。

Linux or Unix

```
sudo systemctl stop greengrass
```

Windows Command Prompt (CMD)

```
sc stop "greengrass"
```

PowerShell

```
Stop-Service -Name "greengrass"
```

2. 在您的核心裝置上，將 AWS IoT Greengrass 軟體下載至名為 `greengrass-2.12.2.zip` 的檔案。

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. 將 AWS IoT Greengrass Core 軟體解壓縮至裝置上的資料夾。將 *GreengrassInstaller* 取代為您要使用的資料夾。

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. 執行下列命令，以 2.12.2 版的 nucleus JAR 檔案覆寫 2.12.3 版的 Greengrass JAR 檔案。

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. 執行下列命令來啟動 Greengrass Core 軟體。

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

如果您在具有預設系統化服務範本的 Linux 裝置上安裝 Greengrass nucleus v2.14.0，您可能會遇到此問題。請執行下列操作：

1. 在 Greengrass 核心裝置上，執行下列命令，將系統化服務檔案還原至 nucleus v2.13.0 及更早版本中的方式。

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. 套用變更。

Linux or Unix

```
sudo systemctl daemon-reload  
sudo systemctl restart greengrass
```

AWS IoT Greengrass 雲端問題

使用以下資訊對 AWS IoT Greengrass 主控台和 API 的問題進行故障診斷。每個項目對應至您在執行動作時可能看到的錯誤訊息。

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

當您從 AWS IoT Greengrass 主控台或使用 [CreateComponentVersion](#) 操作建立元件版本時，您可能會看到此錯誤。

此錯誤表示您的配方不是有效的 JSON 或 YAML。請檢查配方的語法，修正任何語法問題，然後再試一次。您可以使用線上 JSON 或 YAML 語法檢查程式來識別配方中的語法問題。

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

當您從 AWS IoT Greengrass 主控台或使用 [CreateComponentVersion](#) 操作建立元件版本時，您可能會看到此錯誤。此錯誤表示元件配方中的 S3 成品無效。

請執行下列操作：

- 檢查 S3 儲存貯體是否位於您建立元件 AWS 區域的相同位置。AWS IoT Greengrass 不支援元件成品的跨區域請求。
- 檢查成品 URI 是否為有效的 S3 物件 URL，並檢查該 S3 物件 URL 上是否存在成品。
- 檢查您的 是否 AWS 帳戶 具有在其 S3 物件 URL 存取成品的許可。

INACTIVE deployment status

當您呼叫 [ListDeployments](#) API 而沒有所需的相依 AWS IoT 政策時，您可能會取得 INACTIVE 部署狀態。您必須擁有必要的許可，才能取得準確的部署狀態。您可以在 [定義的動作 AWS IoT Greengrass V2](#) 中尋找相依動作，並遵循 所需的許可 ListDeployments。如果沒有必要的相依 AWS IoT 許可，您仍然會看到部署狀態，但您可能會看到不正確的部署狀態 INACTIVE。

核心裝置部署問題

針對 Greengrass 核心裝置上的部署問題進行故障診斷。每個項目對應至您在核心裝置上可能看到的日誌訊息。

主題

- [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.PackageDownloadException`
Failed to download artifact

當 AWS IoT Greengrass 核心裝置套用部署時，當核心軟體無法下載元件成品時，您可能會看到此錯誤。部署會因為此錯誤而失敗。

當您收到此錯誤時，日誌也包含堆疊追蹤，可用來識別特定問題。下列每個項目對應至您在Failed to download artifact錯誤訊息的堆疊追蹤中可能看到的訊息。

主題

- [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, ...)

[PackageDownloadException 錯誤](#)可能會在下列情況下包含此堆疊追蹤：

- 元件成品不適用於您在元件配方中指定的 S3 物件 URL。確認您已將成品上傳到 S3 儲存貯體，且成品 URI 符合儲存貯體中成品的 S3 物件 URL。
- 核心裝置的[字符交換角色](#)不允許 AWS IoT Greengrass Core 軟體從您在元件配方中指定的 S3 物件 URL 下載元件成品。檢查字符交換角色s3:GetObject是否允許使用成品所在的 S3 物件 URL。

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>)

當核心裝置沒有呼叫的許可時，[PackageDownloadException 錯誤](#)可能包含此堆疊追蹤s3:GetBucketLocation。錯誤訊息也包含下列訊息。

```
reason: Failed to determine S3 bucket location
```

檢查核心裝置的[字符交換角色](#)s3:GetBucketLocation是否允許提供成品的 S3 儲存貯體。

Error:

com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.

當 AWS IoT Greengrass 核心裝置套用部署時，當核心軟體無法下載元件成品時，您可能會看到此錯誤。部署失敗，因為下載的成品檔案檢查總和與建立元件時 AWS IoT Greengrass 計算的檢查總和不相符。

請執行下列操作：

- 檢查成品檔案是否在您託管的 S3 儲存貯體中變更。如果檔案在您建立元件後發生變更，請將其還原至核心裝置預期的先前版本。如果您無法將檔案還原至先前的版本，或想要使用新版本的檔案，請使用成品檔案建立新的元件版本。

- 檢查核心裝置的網際網路連線。如果成品檔案在下載時損毀，可能會發生此錯誤。建立新的部署，然後再試一次。

Error:

`com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException`
Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

當核心裝置找不到符合該核心裝置部署需求的元件版本時，您可能會看到此錯誤。核心裝置會檢查 AWS IoT Greengrass 服務和本機裝置上的元件。錯誤訊息包含每個部署的目標，以及元件的部署版本需求。部署目標可以是物件、物件群組或 `LOCAL_DEPLOYMENT`，代表核心裝置上的本機部署。

此問題可能發生在下列情況下：

- 核心裝置是具有衝突元件版本需求的多個部署的目標。例如，核心裝置可能是包含 `com.example.HelloWorld` 元件的多個部署的目標，其中一個部署需要 1.0.0 版，另一個則需要 1.0.1 版。不可能有一個符合這兩個需求的元件，因此部署會失敗。
- 元件版本不存在於 AWS IoT Greengrass 服務或本機裝置上。例如，元件可能已刪除。
- 存在符合版本需求的元件版本，但沒有任何版本與核心裝置的平台相容。
- 核心裝置的 AWS IoT 政策不會授予 `greengrass:ResolveComponentCandidates` 許可。在錯誤日誌 Status Code: 403 中尋找以識別此問題。若要解決此問題，請將 `greengrass:ResolveComponentCandidates` 許可新增至核心裝置的 AWS IoT 政策。如需詳細資訊，請參閱 [AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#)。

若要解決此問題，請修改部署以包含相容的元件版本或移除不相容的元件版本。如需如何修訂雲端部署的詳細資訊，請參閱 [修訂部署](#)。如需如何修訂本機部署的詳細資訊，請參閱 [AWS IoT Greengrass CLI 部署建立](#) 命令。

`software.amazon.awssdk.services.greengrassv2.data.model.ResourceNotFoundException`
The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

當您將元件部署到核心裝置時，您可能會看到此錯誤，而且元件不會列出與核心裝置平台相容的平台。執行以下任意一項：

- 如果元件是自訂 Greengrass 元件，您可以更新元件，使其與核心裝置相容。新增符合核心裝置平台的新資訊清單，或更新現有資訊清單以符合核心裝置的平台。如需詳細資訊，請參閱[AWS IoT Greengrass 元件配方參考](#)。
- 如果元件是由 提供 AWS，請檢查另一個版本的元件是否與核心裝置相容。如果沒有相容版本，[AWS re:Post](#)請使用 [AWS IoT Greengrass 標籤](#)聯絡我們，或聯絡 [支援](#)。

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

當您部署相依於 [Greengrass 核的](#) 元件時，您可能會看到此錯誤，而核心裝置會執行比最新可用次要版本更舊的 Greengrass 核版本。發生此錯誤是因為 AWS IoT Greengrass Core 軟體嘗試自動將元件更新至最新的相容版本。不過，AWS IoT Greengrass 核心軟體可防止 Greengrass 核更新為新的次要版本，因為數個 AWS 提供的元件取決於 Greengrass 核的特定次要版本。如需詳細資訊，請參閱[Greengrass 核更新行為](#)。

您必須[修改部署](#)，以指定要使用的 Greengrass 核版本。執行以下任意一項：

- 修改部署以指定核心裝置目前執行的 Greengrass 核版本。
- 修改部署以指定 Greengrass 核的較新次要版本。如果您選擇此選項，您還必須更新所有 AWS 提供的元件版本，這些元件取決於 Greengrass 核的特定次要版本。如需詳細資訊，請參閱[AWS 提供的元件](#)。

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

當您將 Greengrass 裝置從一個物件群組移至另一個物件群組，然後使用需要 Greengrass 重新啟動的部署返回原始群組時，您可能會看到此錯誤。

若要解決此問題，請重新建立裝置的啟動目錄。我們也強烈建議升級至 2.9.6 版或更新版本的 Greengrass 核。

以下是用於重新建立啟動目錄的 Linux 指令碼。將指令碼儲存在名為 `fix_directory.sh` 的檔案中。

```
#!/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
```

若要執行指令碼，請執行下列命令：

```
[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.RetryableDeploymentDocumentDownloadException`

Greengrass Cloud Service returned an error when getting full deployment configuration

當核心裝置收到大型部署文件時，您可能會看到此錯誤，這是大於 7 KB（針對目標物件的部署）或 31 KB（針對目標物件群組的部署）的部署文件。若要擷取大型部署文件，核心裝置的 AWS IoT 政策必須允許 `greengrass:GetDeploymentConfiguration` 許可。當核心裝置沒有此許可時，可能會發生此錯誤。當此錯誤發生時，部署會無限期重試，且其狀態為進行中 (IN_PROGRESS)。

若要解決此問題，請將 `greengrass:GetDeploymentConfiguration` 許可新增至核心裝置的 AWS IoT 政策。如需詳細資訊，請參閱 [更新核心裝置的 AWS IoT 政策](#)。

Warn: `com.aws.greengrass.deployment.DeploymentService`: Failed to get thing group hierarchy

當核心裝置收到部署且核心裝置 AWS IoT 的政策不允許 `greengrass:ListThingGroupsForCoreDevice` 許可時，您可能會看到此警告。當您建立部署時，核心裝置會使用此許可來識別其物件群組，並移除您從中移除核心裝置之任何物件群組的元件。如果核心裝置執行 [Greengrass 核 v2.5.0](#)，部署會失敗。如果核心裝置執行 Greengrass 核 v2.5.1 或更新版本，部署會繼續進行，但不會移除元件。如需物件群組移除行為的詳細資訊，請參閱 [將 AWS IoT Greengrass 元件部署至裝置](#)。

若要更新核心裝置的行為以移除您移除核心裝置之物件群組的元件，請將 `greengrass:ListThingGroupsForCoreDevice` 許可新增至核心裝置 AWS IoT 的政策。如需詳細資訊，請參閱 [更新核心裝置的 AWS IoT 政策](#)。

Info: `com.aws.greengrass.deployment.DeploymentDocumentDownloader`: Calling Greengrass cloud to get full deployment configuration

您可能會在未發生錯誤的情況下多次看到此資訊訊息，因為核心裝置會在 DEBUG 日誌層級記錄錯誤。當核心裝置收到大型部署文件時，可能會發生此問題。發生此問題時，部署會無限期重試，且其狀態為進行中 (IN_PROGRESS)。如需如何解決此問題的詳細資訊，請參閱 [此故障診斷項目](#)。

Caused by:

software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

當資料平面 API 沒有 `iot:Connect` 許可時，您可能會看到此錯誤。如果您沒有正確的政策，您將會收到 `GreengrassV2DataException: 403`。若要建立許可政策，請遵循下列指示：[建立 AWS IoT 政策](#)。

核心裝置元件問題

針對核心裝置上的 Greengrass 元件問題進行故障診斷。

主題

- [Warn: '<command>' is not recognized as an internal or external command](#)
- [Python 指令碼不會記錄訊息](#)
- [變更預設組態時，元件組態不會更新](#)
- [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

當 AWS IoT Greengrass 核心軟體無法在元件的生命週期指令碼中執行命令時，您可能會在 Greengrass 元件的日誌中看到此錯誤。由於此錯誤BROKEN，元件的狀態會變成。如果執行元件的系統使用者，例如 ggc_user，在 [PATH](#) 的資料夾中找不到命令的可執行檔，則可能會發生此錯誤。

在 Windows 裝置上，檢查包含可執行檔的資料夾是否位於執行元件之PATH系統使用者的 中。如果遺失，PATH請執行下列其中一項操作：

- 將可執行檔的資料夾新增至可供所有使用者使用的 PATH 系統變數。然後，重新啟動元件。

如果您執行 Greengrass 核 2.5.0，在更新PATH系統變數之後，您必須重新啟動 AWS IoT Greengrass Core 軟體，以使用更新的 執行元件PATH。如果 AWS IoT Greengrass 核心軟體在您重新啟動軟體PATH後未使用更新的，請重新啟動裝置，然後再試一次。如需詳細資訊，請參閱[執行 AWS IoT Greengrass 核心軟體](#)。

- 將可執行檔的資料夾新增至執行元件之系統使用者PATH的使用者變數。

Python 指令碼不會記錄訊息

Greengrass 核心裝置會收集日誌，供您用來識別元件的問題。如果您的 Python 指令碼stdout和stderr訊息未出現在元件日誌中，您可能需要在 Python 中清除緩衝或停用這些標準輸出串流的緩衝。執行下列任何一項：

- 使用 [-u](#) 引數執行 Python，以在 stdout和 上停用緩衝stderr。

Linux or Unix

```
python3 -u hello_world.py
```

Windows

```
py -3 -u hello_world.py
```

- 在元件的配方中使用 [Setenv](#)，將 [PYTHONUNBUFFERED](#) 環境變數設定為非空白字串。此環境變數會在 stdout和 上停用緩衝stderr。
- 排清 stdout或 stderr 串流的緩衝區。執行以下任意一項：

- 列印時排清訊息。

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

- 列印後排清訊息。您可以在排清串流之前傳送多個訊息。

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

如需如何驗證 Python 指令碼輸出日誌訊息的詳細資訊，請參閱 [監控 AWS IoT Greengrass 日誌](#)。

變更預設組態時，元件組態不會更新

當您在元件的配方DefaultConfiguration中變更時，新的預設組態不會在部署期間取代元件的現有組態。若要套用新的預設組態，您必須將元件的組態重設為其預設設定。部署元件時，請指定單一空字串做為[重設更新](#)。

Console

重設路徑

```
[""]
```

AWS CLI

下列命令會建立核心裝置的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-deployment.json
```

reset-configuration-deployment.json 檔案包含下列 JSON 文件。

```
{
  "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

下列 [Greengrass CLI](#) 命令會在核心裝置上建立本機部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

reset-configuration-deployment.json 檔案包含下列 JSON 文件。

```
{
  "com.example.HelloWorld": {
    "RESET": [""],
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

當元件沒有對資源執行 IPC 操作的許可時，您可能會在 Greengrass 元件的日誌中看到此錯誤。若要授予元件呼叫 IPC 操作的許可，請在元件的組態中定義 IPC 授權政策。如需詳細資訊，請參閱[授權元件執行 IPC 操作](#)。

Tip

如果您在元件的配方DefaultConfiguration中變更，則必須將元件的組態重設為其新的預設組態。部署元件時，請指定單一空白字串做為[重設更新](#)。如需詳細資訊，請參閱[變更預設組態時，元件組態不會更新](#)。

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

如果多個 IPC 授權政策，包括核心裝置上的所有元件，使用相同的政策 ID，您可能會看到此錯誤。

請檢查元件的 IPC 授權政策、修正任何重複項目，然後再試一次。若要建立唯一的政策 IDs，建議您結合元件名稱、IPC 服務名稱和計數器。如需詳細資訊，請參閱[授權元件執行 IPC 操作](#)。

Tip

如果您在元件的配方DefaultConfiguration中變更，則必須將元件的組態重設為其新的預設組態。部署元件時，請指定單一空白字串做為[重設更新](#)。如需詳細資訊，請參閱[變更預設組態時，元件組態不會更新](#)。

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

當核心裝置無法從[字符交換服務](#)取得 AWS 登入資料時，您可能會看到此錯誤。HTTP 400 狀態碼表示此錯誤是因為核心裝置的[字符交換 IAM 角色](#)不存在，或沒有允許 AWS IoT 登入資料提供者擔任的信任關係。

請執行下列操作：

1. 識別核心裝置使用的字符交換角色。錯誤訊息包含核心裝置 AWS IoT 的角色別名，其指向權杖交換角色。在開發電腦上執行下列命令，並將 *MyGreengrassCoreTokenExchangeRoleAlias* 取代為錯誤訊息中 AWS IoT 角色別名的名稱。

```
aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
```

回應包含字符交換 IAM 角色的 Amazon Resource Name (ARN)。

```
{
  "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. 檢查角色是否存在。執行下列命令，並以字交換角色的名稱取代 *MyGreengrassV2TokenExchangeRole*。

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

如果命令傳回 `NoSuchEntity` 錯誤，則角色不存在，您必須建立該角色。如需如何建立和設定此角色的詳細資訊，請參閱 [授權核心裝置與 AWS 服務互動](#)。

3. 檢查角色是否具有信任關係，允許 AWS IoT 登入資料提供者擔任該角色。上一個步驟的回應包含 `AssumeRolePolicyDocument`，定義角色的信任關係。角色必須定義允許 `credentials.iot.amazonaws.com` 擔任的信任關係。本文件看起來應該類似下列範例。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

如果角色的信任關係不允許 `credentials.iot.amazonaws.com` 擔任，您必須將此信任關係新增至角色。如需更多資訊，請參閱 AWS Identity and Access Management IAM 使用者指南中的 [修改角色](#)。

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

當核心裝置無法從[字符交換服務](#)取得 AWS 登入資料時，您可能會看到此錯誤。HTTP 403 狀態碼表示發生此錯誤是因為核心裝置 AWS IoT 的政策未授予核心裝置 AWS IoT 角色別名的 `iot:AssumeRoleWithCertificate` 許可。

檢閱核心裝置 AWS IoT 的政策，並新增核心裝置 AWS IoT 角色別名的 `iot:AssumeRoleWithCertificate` 許可。錯誤訊息包含核心裝置目前的 AWS IoT 角色別名。如需此許可以及如何更新核心裝置 AWS IoT 政策的詳細資訊，請參閱 [AWS IoT Greengrass V2 核心裝置的最低 AWS IoT 政策](#) 和 [更新核心裝置的 AWS IoT 政策](#)。

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

當元件嘗試請求 AWS 登入資料且無法連線至[字符交換服務](#)時，您可能會看到此錯誤。

請執行下列操作：

- 檢查元件是否宣告對字符交換服務元件的相依性 `aws.greengrass.TokenExchangeService`。如果沒有，請新增相依性並重新部署元件。
- 如果元件在 Docker 中執行，請確定您根據套用的正確網路設定和環境變數在 [Docker 容器元件 \(Linux\) 中使用 AWS 登入資料](#)。
- 如果元件是以 NodeJS 撰寫，請將 [dns.setDefaultResultOrder](#) 設定為 `ipv4first`。
- 檢查 `/etc/hosts` 是否有開頭為 `::1` 且包含的項目 `localhost`。移除項目，以查看是否導致元件連接到錯誤地址的字符交換服務。

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

當元件未執行[字符交換服務](#)，且元件嘗試請求 AWS 登入資料時，您可能會看到此錯誤。

請執行下列操作：

- 檢查元件是否宣告對字符交換服務元件的相依性 `aws.greengrass.TokenExchangeService`。如果沒有，請新增相依性並重新部署元件。

- 檢查元件是否在其install生命週期中使用 AWS 登入資料。AWS IoT Greengrass 在install生命週期期間，不保證字符交換服務的可用性。更新元件以將使用 AWS 登入資料之程式碼移至 startup 或 run 生命週期，然後重新部署元件。

copyFrom: <configurationPath> is already a container, not a leaf

當您將組態值從容器類型（清單或物件）變更為非容器類型（字串、數字或布林值）時，您可能會看到此錯誤。請執行下列操作：

1. 檢查元件的配方，查看其預設組態是否將該組態值設定為清單或物件。若是如此，請移除或變更該組態值。
2. 建立部署，將該組態值重設為其預設值。如需詳細資訊，請參閱[建立部署](#)及[更新元件組態](#)。

然後，您可以將該組態值設定為字串、數字或布林值。

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginError logging into the registry using credentials - 'The stub received bad data.'

當 [Docker 應用程式管理員元件](#) 嘗試從 Amazon Elastic Container Registry (Amazon ECR) 中的私有儲存庫下載 Docker 映像時，您可能會在 Greengrass 核日誌中看到此錯誤。如果您使用 Docker wincred 登入資料協助程式 ()，就會發生此錯誤 docker-credential-wincred。 <https://github.com/docker/docker-credential-helpers> 因此，Amazon ECR 無法存放登入憑證。

採取下列其中一個動作：

- 如果您不使用 wincred Docker 登入資料協助程式，請從核心裝置移除 docker-credential-wincred 程式。
- 如果您使用 wincred Docker 登入資料協助程式，請執行下列動作：
 1. 在核心裝置上重新命名 docker-credential-wincred 程式。wincred 將取代為 Windows Docker 登入資料協助程式的新名稱。例如，您可以將它重新命名為 docker-credential-wincredreal。
 2. 更新 Docker 組態檔案 (.docker/config.json) 中的 credsStore 選項，以使用 Windows Docker 登入資料協助程式的新名稱。例如，如果您將程式重新命名為 docker-credential-wincredreal，請將 credsStore 選項更新為 wincredreal。

```
{  
  "credsStore": "wincredreal"  
}
```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

當執行元件程序的系統使用者有過期的密碼時，您可能會在 Windows 核心裝置上看到此錯誤 `ggc_user`，例如。因此，AWS IoT Greengrass Core 軟體無法以該系統使用者身分執行元件程序。

更新 Greengrass 系統使用者的密碼

1. 以管理員身分執行下列命令，以設定使用者的密碼。將 `ggc_user` 取代為系統使用者，並將 `##` 取代為要設定的密碼。

```
net user ggc_user password
```

2. 使用 [PsExec 公用程式](#)，將使用者的新密碼存放在 LocalSystem 帳戶的 Credential Manager 執行個體中。將 `##` 取代為您設定的使用者密碼。

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

Tip

根據您的 Windows 組態，使用者的密碼可能會設定為在未來的日期過期。為了確保您的 Greengrass 應用程式持續運作，請追蹤密碼過期的時間，並在過期之前更新它。您也可以將使用者的密碼設定為永不過期。

- 若要檢查使用者及其密碼何時過期，請執行下列命令。

```
net user ggc_user | findstr /C:expires
```

- 若要將使用者的密碼設定為永不過期，請執行下列命令。

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- 如果您使用的是 [wmic 已棄用命令](#) 的 Windows 10 或更新版本，請執行下列 PowerShell 命令。

```
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

當您將串流管理員 v2.0.7 升級至 v2.0.8 和 v2.0.11 之間的版本時，如果元件無法啟動，您可能會在串流管理員元件的日誌中看到下列錯誤。

```
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}
```

如果您部署了串流管理員 v2.0.7，並且想要升級到更新版本，則必須直接升級到串流管理員 v2.0.12。如需串流管理員元件的詳細資訊，請參閱 [串流管理員](#)。

核心裝置 Lambda 函數元件問題

對核心裝置上的 Lambda 函數元件問題進行故障診斷。

主題

- [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

在下列情況下，當您執行容器化 Lambda 函數時，可能會看到此錯誤：

- 核心裝置未針對記憶體或裝置 cgroup 啟用 cgroup v1。
- 核心裝置已啟用 cgroups v2。Greengrass Lambda 函數需要 cgroups v1，而 cgroups v1 和 v2 是互斥的。

若要啟用 cgroups v1，請使用下列 Linux 核心參數啟動裝置。

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

Tip

在 Raspberry Pi 上，編輯 `/boot/cmdline.txt` 檔案以設定裝置的核心參數。

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>

當您在 V2 核心裝置上執行使用核心 SDK 的 V1 Lambda 函數時，您可能會看到此錯誤，而無需在[舊版訂閱路由器元件中指定訂閱](#)。AWS IoT Greengrass V2 若要修正此問題，請部署並設定舊版訂閱路由器以指定必要的訂閱。如需詳細資訊，請參閱[匯入 V1 Lambda 函數](#)。

元件版本已停止

當核心裝置上的元件版本停止時，您可能會在個人運作狀態儀表板 (PHD) 上看到通知。元件版本會在停止後 60 分鐘內將此通知傳送至您的 PHD。

若要查看您需要修改的部署，請使用執行下列動作 AWS Command Line Interface：

1. 執行下列命令以取得核心裝置的清單。

```
aws greengrassv2 list-core-devices
```

2. 執行下列命令，從步驟 1 擷取每個核心裝置上的元件狀態。`coreDeviceName` 將取代為要查詢的每個核心裝置的名稱。

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

3. 使用從先前步驟安裝的已停止元件版本來收集核心裝置。
4. 執行下列命令，從步驟 3 擷取每個核心裝置的所有部署任務狀態。*coreDeviceName* 將取代為要查詢的核心裝置名稱。

```
aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
```

回應包含核心裝置的部署任務清單。您可以修改部署，以選擇另一個元件版本。如需如何修訂部署的詳細資訊，請參閱[修訂部署](#)。

Greengrass 命令列界面問題

針對 [Greengrass CLI](#) 的問題進行故障診斷。

主題

- [java.lang.RuntimeException: Unable to create ipc client](#)

java.lang.RuntimeException: Unable to create ipc client

當您執行 Greengrass CLI 命令，並指定與安裝 AWS IoT Greengrass Core 軟體不同的根資料夾時，您可能會看到此錯誤。

執行下列其中一項操作來設定根路徑，並將 */greengrass/v2* 取代為 AWS IoT Greengrass Core 軟體安裝的路徑：

- 將 GGC_ROOT_PATH 環境變數設為 */greengrass/v2*。
- 將 `--ggcRootPath` */greengrass/v2* 引數新增至命令，如下列範例所示。

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface 問題

針對 AWS CLI 的問題進行故障診斷 AWS IoT Greengrass V2。

主題

- [Error: Invalid choice: 'greengrassv2'](#)

Error: Invalid choice: 'greengrassv2'

當您使用 AWS CLI（例如，）執行 AWS IoT Greengrass V2 命令時，您可能會看到此錯誤 `aws greengrassv2 list-core-devices`。

此錯誤表示您有不支援的 AWS CLI 版本 AWS IoT Greengrass V2。若要 AWS IoT Greengrass V2 搭配使用 AWS CLI，您必須具有下列其中一個版本或更新版本：

- Minimum AWS CLI V1 版本：v1.18.197
- Minimum AWS CLI V2 版本：v2.1.11

Tip

您可以執行下列命令來檢查 AWS CLI 您擁有的 版本。

```
aws --version
```

若要解決此問題，請將 更新 AWS CLI 為 支援的更新版本 AWS IoT Greengrass V2。如需詳細資訊，請參閱 AWS Command Line Interface 《使用者指南》中的 [安裝、更新和解除安裝 AWS CLI](#)。

詳細部署錯誤代碼

使用這些區段中的錯誤代碼和解決方案，以協助解決使用 Greengrass 核 2.8.0 版或更新版本時元件部署的問題。

Greengrass 核會將部署錯誤報告為從最不特定到最特定可用程式碼的階層。您可以使用此階層來協助找出部署錯誤的原因。例如，下列是可能的錯誤階層：

- DEPLOYMENT_FAILURE
 - ARTIFACT_DOWNLOAD_ERROR
 - IO_ERROR
 - DISK_SPACE_CRITICAL

錯誤代碼會依類型整理。每種類型都代表可能發生的錯誤類別。會在主控台、API 和中 AWS IoT Greengrass 報告這些錯誤類型 AWS CLI。根據錯誤階層中報告的錯誤，可能會有多個錯誤類型。針對上述範例，傳回的錯誤類型為 `DEVICE_ERROR`。

類型為：

- `PERMISSION_ERROR` – 存取需要許可的操作遭拒。
- `REQUEST_ERROR` – 由於部署文件中的問題而發生錯誤。
- `COMPONENT_RECIPE_ERROR` – 由於元件配方中的問題而發生錯誤。
- `AWS_COMPONENT_ERROR` – 啟動或移除 AWS 提供的元件時發生錯誤。
- `USER_COMPONENT_ERROR` – 啟動或移除使用者元件時發生錯誤。
- `COMPONENT_ERROR` – 啟動或移除元件時發生錯誤，但 Greengrass 核無法判斷元件是 AWS 提供的元件還是使用者元件。
- `DEVICE_ERROR` – 本機 I/O 發生錯誤或發生另一個裝置錯誤。
- `DEPENDENCY_ERROR` – 部署無法從 Amazon S3 下載成品，或無法從 ECR 登錄檔提取映像。
- `HTTP_ERROR` – HTTP 請求發生錯誤。
- `NETWORK_ERROR` – 裝置網路發生錯誤。
- `NUCLEUS_ERROR` – Greengrass 核找不到元件或找不到作用中的核版本。
- `SERVER_ERROR` – 伺服器傳回 500 錯誤以回應請求。
- `CLOUD_SERVICE_ERROR` – AWS IoT Greengrass 雲端服務發生錯誤。
- `UNKNOWN_ERROR` – 元件擲回未檢查的例外狀況。

本節中的許多錯誤會報告 AWS IoT Greengrass 核心日誌中的其他資訊。這些日誌存放在核心裝置的本機檔案系統上。AWS IoT Greengrass 核心軟體和每個個別元件都有日誌。如需存取日誌的資訊，請參閱 [存取檔案系統日誌](#)。

許可錯誤

`ACCESS_DENIED`

當 AWS 服務操作傳回 403 錯誤時，您可能會收到此錯誤，因為未正確設定許可。如需詳細資訊，請檢查更具體的錯誤代碼。

`GET_DEPLOYMENT_CONFIGURATION_ACCESS_DENIED`

當 AWS IoT 政策不允許呼叫 `GetDeploymentConfiguration` 操作的許可時，您可能會收到此錯誤。將 `greengrass::GetDeploymentConfiguration` 許可新增至核心裝置的政策。

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

當核心裝置 AWS IoT 政策不允許 `greengrass:GetComponentVersionArtifact` 許可時，您可能會收到此錯誤。將許可新增至核心裝置的政策。

RESOLVE_COMPONENT_CANDIDATES_ACCESS_DENIED

當核心裝置 AWS IoT 政策不允許 `greengrass:ResolveComponentCandidates` 許可時，您可能會收到此錯誤。將許可新增至核心裝置的政策。

GET_ECR_CREDENTIAL_ERROR

當部署無法透過 ECR 中的私有登錄檔進行身分驗證時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後再次嘗試部署。

USER_NOT_AUTHORIZED_FOR_DOCKER

當 Greengrass 使用者未獲授權使用 Docker 時，您可能會收到此錯誤。請確定您以根身分執行 Greengrass，或使用者已新增至 `docker` 群組。然後再次嘗試部署。

S3_ACCESS_DENIED

當 Amazon S3 操作傳回 403 錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

S3_HEAD_OBJECT_ACCESS_DENIED

當裝置的字符交換角色不允許 AWS IoT Greengrass Core 軟體從您在元件配方中指定的 S3 物件 URL 下載元件成品，或元件成品無法使用時，您可能會收到此錯誤。檢查權杖交換角色 `s3:GetObject` 是否允許使用成品可用的 S3 物件 URL，以及成品是否存在。

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

當裝置的字符交換角色不允許提供成品的 Amazon S3 儲存貯體的 `s3:GetBucketLocation` 許可時，您可能會收到此錯誤。檢查裝置的是否允許許可，然後再次嘗試部署。

S3_GET_OBJECT_ACCESS_DENIED

當裝置的字符交換角色不允許 AWS IoT Greengrass Core 軟體從您在元件配方中指定的 S3 物件 URL 下載元件成品，或元件成品無法使用時，您可能會收到此錯誤。檢查權杖交換角色 `s3:GetObject` 是否允許使用成品可用的 S3 物件 URL，以及成品是否存在。

請求錯誤

NUCLEUS_MISSING_REQUIRED_CAPABILITIES

當部署中的核版本無法執行請求的操作時，您可能會收到此錯誤，例如下載大型組態或設定 Linux 資源限制。使用支援 操作的 nucleus 版本重試部署。

MULTIPLE_NUCLEUS_RESOLVED_ERROR

部署嘗試部署多個核心元件時，您可能會收到此錯誤。檢查日誌以查看錯誤的原因，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡 支援。

COMPONENT_CIRCULAR_DEPENDENCY_ERROR

當部署中的兩個元件彼此相依時，您可能會收到此錯誤。修改元件設定，讓部署中的元件不會彼此依賴。

UNAUTHORIZED_NUCLEUS_MINOR_VERSION_UPDATE

當部署中的元件需要更新核心次要版本，但未在部署中指定該版本時，您可能會收到此錯誤。這有助於減少相依於不同版本的元件意外次要版本更新。在部署中包含新的次要核心版本。

MISSING_DOCKER_APPLICATION_MANAGER

當您在不部署 Docker 應用程式管理員的情況下部署 Docker 元件時，可能會收到此錯誤。請確定您的部署包含 Docker 應用程式管理員。

MISSING_TOKEN_EXCHANGE_SERVICE

當部署想要從私有 ECR 登錄檔下載 Docker 映像成品而不部署字符交換服務時，您可能會收到此錯誤。請確定您的部署包含權杖交換服務。

COMPONENT_VERSION_REQUIREMENTS_NOT_MET

當版本限制衝突或元件版本不存在時，您可能會收到此錯誤。如需詳細資訊，請參閱[Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>](https://docs.aws.amazon.com/greengrass/componentmanager/exceptions/NoAvailableComponentVersionException/Failed-to-negotiate-component-version-with-cloud-and-no-local-applicable-version-satisfying-requirement)。

THROTTLING_ERROR

當 AWS 服務操作超過速率配額時，您可能會收到此錯誤。重新嘗試部署。

CONFLICTED_REQUEST

當 AWS 服務操作傳回 409 錯誤時，您可能會收到此錯誤，因為您部署嘗試一次執行多個操作。重新嘗試部署。

RESOURCE_NOT_FOUND

當 AWS 服務操作傳回 404 錯誤時，您可能會收到此錯誤，因為找不到資源。檢查日誌是否有遺失的資源。

RUN_WITH_CONFIG_NOT_VALID

當指定執行元件的 `posixUser`、`posixGroup` 或 `windowsUser` 資訊無效時，您可能會收到此錯誤。檢查使用者是否有效，然後重試部署。

UNSUPPORTED_REGION

當不支援為部署指定的區域時，您可能會收到此錯誤 AWS IoT Greengrass。請檢查區域，然後再試一次部署。

IOT_CRED_ENDPOINT_NOT_VALID

當組態中指定的 AWS IoT 登入資料端點無效時，您可能會收到此錯誤。請檢查端點，然後再試一次您的請求。

IOT_DATA_ENDPOINT_NOT_VALID

當組態中指定的 AWS IoT 資料端點無效時，您可能會收到此錯誤。請檢查端點，然後再試一次您的請求。

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

當您在元件配方中指定的 S3 物件 URL 無法使用元件成品時，您可能會收到此錯誤。確認您已將成品上傳到 S3 儲存貯體，且成品 URI 符合儲存貯體中成品的 S3 物件 URL。

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

找不到 Amazon S3 儲存貯體時，您可能會收到此錯誤。檢查儲存貯體是否存在，然後重試部署。

S3_GET_OBJECT_RESOURCE_NOT_FOUND

當您在元件配方中指定的 S3 物件 URL 無法使用元件成品時，您可能會收到此錯誤。確認您已將成品上傳到 S3 儲存貯體，且成品 URI 符合儲存貯體中成品的 S3 物件 URL。

IO_MAPPING_ERROR

當剖析部署文件或配方時發生 I/O 錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

元件配方錯誤

RECIPE_PARSE_ERROR

當因為配方的結構發生錯誤而無法剖析部署配方時，您可能會收到此錯誤。檢查配方的格式是否正確，然後重試部署。

RECIPE_METADATA_PARSE_ERROR

當無法剖析從雲端下載的部署配方中繼資料時，您可能會收到此錯誤。聯絡 支援。

ARTIFACT_URI_NOT_VALID

當配方中的成品 URI 格式不正確時，您可能會收到此錯誤。檢查日誌中是否有無效 URI、更新配方中的 URI，然後再次嘗試部署。

S3_ARTIFACT_URI_NOT_VALID

當配方中成品的 Amazon S3 URI 無效時，您可能會收到此錯誤。檢查日誌中是否有無效 URI、更新配方中的 URI，然後再次嘗試部署。

DOCKER_ARTIFACT_URI_NOT_VALID

當配方中成品的 Docker URI 無效時，您可能會收到此錯誤。檢查日誌中是否有無效 URI、更新配方中的 URI，然後再次嘗試部署。

EMPTY_ARTIFACT_URI

當配方中未指定成品的 URI 時，您可能會收到此錯誤。檢查日誌中是否有缺少 URI 的成品、更新配方中的 URI，然後再次嘗試部署。

EMPTY_ARTIFACT_SCHEME

當未為成品定義 URI 結構描述時，您可能會收到此錯誤。檢查日誌中是否有無效 URI、更新配方中的 URI，然後再次嘗試部署。

UNSUPPORTED_ARTIFACT_SCHEME

當執行中的 nucleus 版本不支援 URI 結構描述時，您可能會收到此錯誤。URI 無效，或者您需要更新 nucleus 版本。如果 URI 無效，請檢查無效 URI 的日誌、更新配方中的 URI，然後再次嘗試部署。

RECIPE_MISSING_MANIFEST

當資訊清單區段未包含在配方中時，您可能會收到此錯誤。將資訊清單新增至配方，然後再次嘗試部署。

RECIPE_MISSING_ARTIFACT_HASH_ALGORITHM

在沒有雜湊演算法的配方中指定非本機成品時，您可能會收到此錯誤。將演算法新增至成品，然後再試一次請求。

ARTIFACT_CHECKSUM_MISMATCH

當下載的成品與配方中指定的摘要不同時，您可能會收到此錯誤。請確定配方包含正確的摘要，然後再次嘗試部署。如需更多資訊，請參閱[Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.](https://docs.aws.amazon.com/greengrass/componentmanager/exceptions/ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.)。

COMPONENT_DEPENDENCY_NOT_VALID

當部署配方中指定的相依性類型無效時，您可能會收到此錯誤。請檢查配方，然後再試一次您的請求。

CONFIG_INTERPOLATE_ERROR

插入配方變數時，您可能會收到此錯誤。如需詳細資訊，請檢查日誌。

IO_MAPPING_ERROR

當剖析部署文件或配方時發生 I/O 錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

AWS 元件錯誤、使用者元件錯誤、元件錯誤

當元件發生問題時，會傳回下列錯誤代碼。報告的實際錯誤類型取決於引發錯誤的特定元件。如果 Greengrass 核將元件識別為提供的元件 AWS IoT Greengrass，則會傳回 `AWS_COMPONENT_ERROR`。如果將元件識別為使用者元件，Greengrass 核會傳回 `USER_COMPONENT_ERROR`。如果 Greengrass 核無法告知，則會傳回 `COMPONENT_ERROR`。

COMPONENT_UPDATE_ERROR

當元件在部署期間未更新時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌，以查看造成錯誤的原因。

COMPONENT_BROKEN

當元件在部署期間損壞時，您可能會收到此錯誤。檢查元件日誌以取得錯誤詳細資訊，然後再次嘗試部署。

REMOVE_COMPONENT_ERROR

當核心無法在部署期間移除元件時，您可能會收到此錯誤。請檢查日誌以取得錯誤詳細資訊，然後再次嘗試部署。

COMPONENT_BOOTSTRAP_TIMEOUT

當元件的引導任務花費超過設定的逾時時間時，您可能會收到此錯誤。增加逾時或縮短引導任務的執行時間，然後再次嘗試部署。

COMPONENT_BOOTSTRAP_ERROR

當元件的引導任務發生錯誤時，您可能會收到此錯誤。請檢查日誌以取得錯誤詳細資訊，然後再次嘗試部署。

COMPONENT_CONFIGURATION_NOT_VALID

當核心無法驗證元件的部署組態時，您可能會收到此錯誤。請檢查日誌以取得錯誤詳細資訊，然後再次嘗試部署。

裝置錯誤

IO_WRITE_ERROR

寫入檔案時，您可能會收到此錯誤。如需詳細資訊，請檢查日誌。

IO_READ_ERROR

從檔案讀取時，您可能會收到此錯誤。如需詳細資訊，請檢查日誌。

DISK_SPACE_CRITICAL

當磁碟空間不足而無法完成部署請求時，您可能會收到此錯誤。您必須擁有至少 20 Mb 的可用空間，或足以容納較大的成品。釋放一些磁碟空間，然後重試部署。

IO_FILE_ATTRIBUTE_ERROR

當無法從檔案系統擷取現有檔案大小時，您可能會收到此錯誤。如需詳細資訊，請檢查日誌。

SET_PERMISSION_ERROR

當無法在下載的成品或成品目錄上設定許可時，您可能會收到此錯誤。如需詳細資訊，請檢查日誌。

IO_UNZIP_ERROR

當成品無法解壓縮時，您可能會收到此錯誤。如需詳細資訊，請檢查日誌。

LOCAL_RECIPE_NOT_FOUND

找不到配方檔案的本機副本時，您可能會收到此錯誤。再次嘗試部署。

LOCAL_RECIPE_CORRUPTED

當配方的本機複本自下載以來已變更時，您可能會收到此錯誤。刪除配方的現有副本，然後重試部署。

LOCAL_RECIPE_METADATA_NOT_FOUND

找不到配方中繼資料檔案的本機副本時，您可能會收到此錯誤。再次嘗試部署。

LAUNCH_DIRECTORY_CORRUPTED

當用於啟動 Greengrass 核的目錄 (/greengrass/v2/alts/current) 自上次啟動核之後已經過修改時，您可能會收到此錯誤。重新啟動核心，然後重試部署。

HASHING_ALGORITHM_UNAVAILABLE

當裝置的 Java 分佈不支援所需的雜湊演算法，或元件配方中指定的雜湊演算法無效時，您可能會收到此錯誤。

DEVICE_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNLOAD

當裝置組態發生錯誤，導致部署無法從 Amazon S3 或 Greengrass 雲端下載成品時，您可能會收到此錯誤。檢查日誌是否有特定的組態錯誤，然後重試部署。

相依性錯誤

DOCKER_ERROR

您可能會在提取 Docker 映像時收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

DOCKER_SERVICE_UNAVAILABLE

當 Greengrass 無法登入 Docker 登錄檔時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後再次嘗試部署。

DOCKER_LOGIN_ERROR

當登入 Docker 時發生意外錯誤時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後再次嘗試部署。

DOCKER_PULL_ERROR

當從登錄檔提取 Docker 映像時發生意外錯誤時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後再次嘗試部署。

DOCKER_IMAGE_NOT_VALID

當請求的 Docker 映像不存在時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後重試部署。

DOCKER_IMAGE_QUERY_ERROR

查詢 Docker 是否有可用的映像時發生非預期的失敗時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後重試部署。

S3_ERROR

下載 Amazon S3 成品時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

S3_RESOURCE_NOT_FOUND

當 Amazon S3 操作傳回 404 錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

S3_BAD_REQUEST

當 Amazon S3 操作傳回 400 錯誤時，您可能會收到此錯誤。檢查日誌是否有特定錯誤，然後再試一次請求。

HTTP 錯誤

HTTP_REQUEST_ERROR

當提出 HTTP 請求時發生錯誤時，您可能會收到此錯誤。檢查日誌是否有特定錯誤。

DOWNLOAD_DEPLOYMENT_DOCUMENT_ERROR

當下載部署文件時發生 HTTP 錯誤時，您可能會收到此錯誤。檢查日誌是否有特定的 HTTP 錯誤。

GET_GREENGRASS_ARTIFACT_SIZE_ERROR

當取得公有元件成品的大小時發生 HTTP 錯誤時，您可能會收到此錯誤。檢查日誌是否有特定的 HTTP 錯誤。

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

當下載公有元件成品時發生 HTTP 錯誤時，您可能會收到此錯誤。檢查日誌是否有特定的 HTTP 錯誤。

網路錯誤

NETWORK_ERROR

在部署期間發生連線問題時，您可能會收到此錯誤。檢查裝置與網際網路的連線，然後重試部署。

Nucleus 錯誤

BAD_REQUEST

當 AWS 雲端操作傳回 400 錯誤時，您可能會收到此錯誤。檢查日誌以查看導致錯誤的 API，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡案例支援。

NUCLEUS_VERSION_NOT_FOUND

當核心裝置找不到作用中核心的版本時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡支援案例。

NUCLEUS_RESTART_FAILURE

當核心在需要重新啟動的部署期間未重新啟動時，您可能會收到此錯誤。檢查載入器日誌以查看錯誤的原因，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡案例支援。

INSTALLED_COMPONENT_NOT_FOUND

當核心找不到已安裝的元件時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡支援案例。

DEPLOYMENT_DOCUMENT_NOT_VALID

當裝置收到無效的部署文件時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌，以查看造成錯誤的原因。

EMPTY_DEPLOYMENT_REQUEST

當裝置收到空的部署請求時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡支援案例。

DEPLOYMENT_DOCUMENT_PARSE_ERROR

當部署請求格式不符合預期的格式時，您可能會收到此錯誤。檢查日誌以查看錯誤的原因，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡支援。

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

當部署請求包含無效的元件中繼資料時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡支援案例。

LAUNCH_DIRECTORY_CORRUPTED

當您將 Greengrass 裝置從一個物件群組移至另一個物件群組，然後使用需要 Greengrass 重新啟動的部署返回原始群組時，您可能會收到此錯誤。若要解決錯誤，請在裝置上重新建立 Greengrass 的啟動目錄。

如需詳細資訊，請參閱[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](#)。

伺服器錯誤

SERVER_ERROR

當 AWS 服務操作傳回 500 錯誤時，您可能會收到此錯誤，因為服務目前無法處理請求。稍後重試部署。

S3_SERVER_ERROR

當 Amazon S3 操作傳回 500 錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

雲端服務錯誤

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

當 Greengrass 雲端服務傳送不相容的回應給 ResolveComponentCandidates 操作時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡案例支援。

DEPLOYMENT_DOCUMENT_SIZE_EXCEEDED

當請求的部署文件超過大小配額上限時，您可能會收到此錯誤。減少部署文件的大小，然後再試一次部署。

GREENGRASS_ARTIFACT_SIZE_NOT_FOUND

當 Greengrass 無法取得公有元件成品的大小時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡案例支援。

DEPLOYMENT_DOCUMENT_NOT_VALID

當裝置收到無效的部署文件時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌，以查看造成錯誤的原因。

EMPTY_DEPLOYMENT_REQUEST

當裝置收到空的部署請求時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡案例支援。

DEPLOYMENT_DOCUMENT_PARSE_ERROR

當部署請求格式不符合預期的格式時，您可能會收到此錯誤。檢查日誌以查看錯誤的原因，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡支援。

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

當部署請求包含無效的元件中繼資料時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡案例支援。

一般錯誤

這些一般錯誤沒有相關聯的錯誤類型。

DEPLOYMENT_INTERRUPTED

當部署因為核心關閉或其他外部事件而無法完成時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

ARTIFACT_DOWNLOAD_ERROR

當下載成品發生問題時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

NO_AVAILABLE_COMPONENT_VERSION

當元件版本不存在於雲端或本機，或存在相依性解析衝突時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

COMPONENT_PACKAGE_LOADING_ERROR

當錯誤處理下載的成品時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

CLOUD_API_ERROR

當呼叫 AWS 服務 API 時發生錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

IO_ERROR

當部署期間發生 I/O 錯誤時，您可能會收到此錯誤。如需詳細資訊，請檢查任何其他錯誤代碼或日誌。

COMPONENT_UPDATE_ERROR

當元件在部署期間未更新時，您可能會收到此錯誤。檢查任何其他錯誤代碼或檢查日誌，以查看造成錯誤的原因。

未知錯誤

DEPLOYMENT_FAILURE

當部署因為擲出未檢查的例外狀況而失敗時，您可能會收到此錯誤。檢查日誌以查看錯誤的原因，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡 支援。

DEPLOYMENT_TYPE_NOT_VALID

當部署類型無效時，您可能會收到此錯誤。檢查日誌以查看造成錯誤的因素，然後檢查 nucleus 軟體更新頁面，查看此問題是否已在更新版本的 nucleus 中修正，或聯絡 案例 支援。

詳細的元件狀態碼

使用這些區段中的狀態碼和解決方案，以協助解決使用 Greengrass nucleus 2.8.0 版或更新版本時元件的問題。

本主題中的許多狀態都會報告 AWS IoT Greengrass 核心日誌中的額外資訊。這些日誌存放在核心裝置的本機檔案系統中。每個個別元件都有日誌。如需存取日誌的資訊，請參閱 [存取檔案系統日誌](#)。

INSTALL_ERROR

當執行安裝指令碼時發生錯誤時，您可能會收到此項目。錯誤碼會在元件日誌中報告。檢查安裝指令碼是否有錯誤，然後再次部署元件。

INSTALL_CONFIG_NOT_VALID

當因為配方的 `install` 區段無效而無法完成元件安裝時，您可能會收到此錯誤。檢查配方的安裝區段是否有錯誤，並再次嘗試部署。

INSTALL_IO_ERROR

安裝元件期間發生 I/O 錯誤時，您可能會收到此訊息。如需錯誤的詳細資訊，請檢查元件錯誤日誌。

INSTALL_MISSING_DEFAULT_RUNWITH

當 AWS IoT Greengrass 無法判斷使用者或群組在安裝元件時使用時，您可能會收到此錯誤。檢查安裝配方的 `runWith` 區段是否包含有效的使用者或群組。

INSTALL_TIMEOUT

當安裝指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加配方 `install` 區段中指定的 `Timeout` 期間，或修改您的安裝指令碼，以在設定的逾時內完成。

STARTUP_ERROR

當執行啟動指令碼時發生錯誤時，您可能會收到此項目。錯誤碼會在元件日誌中報告。檢查安裝指令碼是否有錯誤，然後再次部署元件。

STARTUP_CONFIG_NOT_VALID

當因為配方的 `startup` 區段無效而無法完成元件安裝時，您可能會收到此錯誤。檢查配方的啟動區段是否有錯誤，然後再試一次部署。

STARTUP_IO_ERROR

當元件啟動期間發生 I/O 錯誤時，您可能會收到此訊息。如需錯誤的詳細資訊，請檢查元件錯誤日誌。

STARTUP_MISSING_DEFAULT_RUNWITH

當 AWS IoT Greengrass 無法判斷執行元件時要使用的使用者或群組時，您可能會收到此錯誤。檢查以確認啟動配方的 `runWith` 區段包含有效的使用者或群組。

STARTUP_TIMEOUT

當啟動指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加配方startup區段中指定的Timeout期間，或修改啟動指令碼，以在設定的逾時內完成。

RUN_ERROR

當執行元件指令碼時發生錯誤時，您可能會收到此項目。錯誤碼會在元件日誌中報告。檢查執行指令碼是否有錯誤，然後再次部署您的元件。

RUN_MISSING_DEFAULT_RUNWITH

當 AWS IoT Greengrass 無法判斷執行元件時要使用的使用者或群組時，您可能會收到此錯誤。檢查以確保您執行配方的 runWith區段包含有效的使用者或群組。

RUN_CONFIG_NOT_VALID

當因為配方的 run區段無效而無法執行元件時，您可能會收到此錯誤。檢查配方的執行區段是否有錯誤，並再次嘗試部署。

RUN_IO_ERROR

當元件執行時，您可能會在發生 I/O 錯誤時收到此訊息。如需錯誤的詳細資訊，請檢查元件錯誤日誌。

RUN_TIMEOUT

當執行指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加配方run區段中指定的Timeout期間，或修改您的執行指令碼，以在設定的逾時內完成。

SHUTDOWN_ERROR

當關閉元件指令碼時發生錯誤時，您可能會收到此訊息。錯誤碼會在元件日誌中報告。檢查關機指令碼是否有錯誤，然後再次部署您的元件。

SHUTDOWN_TIMEOUT

當關閉指令碼未在設定的逾時期間內完成時，您可能會收到此錯誤。您可以增加配方shutdown區段中指定的Timeout期間，或修改您的執行指令碼，以在設定的逾時內完成。

標記您的 AWS IoT Greengrass Version 2 資源

使用標籤，您可以在 AWS IoT Greengrass 中組織和管理您的資源。您可以使用標籤將中繼資料指派給資源，也可以在 IAM 政策中使用標籤來定義對資源的條件式存取。

Note

目前，AWS IoT 帳單群組或成本分配報告不支援 Greengrass 資源標籤。

在 中使用標籤 AWS IoT Greengrass V2

您可以使用標籤，AWS IoT Greengrass 依用途、擁有者、環境或任何其他使用案例的分類來分類資源。當您有許多相同類型的資源時，標籤可協助您更輕鬆地識別特定資源。

每個標籤皆包含由您定義的一個金鑰與一個選用值。例如，您可以為核心裝置定義一組標籤，協助您由擁有裝置的客戶追蹤它們。我們建議您為每種資源類型建立符合您需求的一組標籤金鑰。使用一致的標籤鍵組，可讓您的資源管理更輕鬆。

使用 標記 AWS 管理主控台

中的標籤編輯器 AWS 管理主控台 提供集中、統一的方式，讓您為來自所有 AWS 服務的資源建立和管理標籤。如需詳細資訊，請參閱 AWS Resource Groups 《使用者指南》中的 [標籤編輯器](#)。

使用 AWS IoT Greengrass V2 API 標記

您也可以使用 AWS IoT Greengrass V2 API 來處理標籤。在建立標籤之前，請注意下列標籤限制。如需詳細資訊，請參閱《》中的 [標籤命名和使用慣例](#) AWS 一般參考。

- 若要在建立資源時新增標籤，請在資源的 tags 屬性中定義標籤。
- 若要將標籤新增至現有資源或更新標籤值，請使用 [TagResource](#) 操作。
- 若要從資源移除標籤，請使用 [UntagResource](#) 操作。
- 若要擷取與資源建立關聯的標籤，請使用 [ListTagsForResource](#) 操作，或描述資源並檢查其 tags 屬性。

下表列出您可以使用 AWS IoT Greengrass V2 API 及其對應的 Create 和 Describe 或 Get 操作來標記的資源。

可標記 AWS IoT Greengrass V2 的資源

資源	建立操作	描述或取得操作
核心裝置	無。在裝置上執行 AWS IoT Greengrass Core 軟體以建立核心裝置。	GetCoreDevice
元件	CreateComponentVersion	DescribeComponent 、 GetComponent
部署	CreateDeployment	GetDeployment

使用下列操作檢視並管理支援標記之資源的標籤：

- [TagResource](#) – 將標籤新增至資源，或更新現有標籤的值。
- [ListTagsForResource](#) – 列出資源的標籤。
- [UntagResource](#) – 從資源移除標籤。

您可以隨時新增或移除資源的標籤。若要變更標籤金鑰的值，請在資源加入定義相同金鑰和新數值的標籤。新值會取代先前的值。您可以將數值設為空白字串，但您無法將數值設為 null。

在您刪除資源時，也會刪除與該資源建立關聯的標籤。

搭配 IAM 政策使用標籤

在您的 IAM 政策中，您可以使用資源標籤來控制使用者存取和許可。例如，政策可讓使用者僅能建立具有特定標籤的資源。政策也可以限制使用者建立或修改具有特定標籤的資源。

Note

如果您使用標籤來允許或拒絕使用者存取資源，您應該拒絕使用者為相同資源新增或移除這些標籤的能力。否則，使用者可能透過修改標籤來避開您的限制，並取得資源的存取。

您可以在政策陳述式的 Condition 元素中使用下列條件內容索引鍵和值，也稱為 Condition 區塊。

`greengrassv2:ResourceTag/tag-key: tag-value`

允許或拒絕資源對具有特定標籤之資源的動作。

`aws:RequestTag/tag-key: tag-value`

建立或修改可標記資源時，需要使用或不使用特定標籤。

`aws:TagKeys: [tag-key, ...]`

建立或修改可標記資源時，需要使用或不使用一組特定的標籤金鑰。

 Note

IAM 政策中的條件內容索引鍵和值僅適用於具有可標記資源作為必要參數的動作。例如，您可以為 [GetComponent](#) 設定標籤型條件式存取。

如需詳細資訊，請參閱《IAM 使用者指南》中的[使用 AWS 資源標籤和 IAM JSON 政策參考控制對資源的存取](#)。 https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies.html

使用 建立 AWS IoT Greengrass 資源 AWS CloudFormation

AWS IoT Greengrass 已與 整合 AWS CloudFormation，此服務可協助您模型化和設定 AWS 資源，以減少建立和管理資源和基礎設施的時間。您可以建立範本來描述您想要的所有 AWS 資源（例如元件版本和部署），並為您 CloudFormation 佈建和設定這些資源。

使用 時 CloudFormation，您可以重複使用範本來一致且重複地設定 AWS IoT Greengrass 資源。描述您的資源一次，然後在多個 AWS 帳戶 和 區域中逐一佈建相同的資源。

AWS IoT Greengrass 和 CloudFormation 範本

若要佈建和設定 AWS IoT Greengrass 和相關服務的資源，您必須了解 [CloudFormation 範本](#)。範本是以 JSON 或 YAML 格式化的文本檔案。這些範本說明您要在 CloudFormation 堆疊中佈建的資源。如果您不熟悉 JSON 或 YAML，您可以使用 CloudFormation 設計工具來協助您開始使用 CloudFormation 範本。如需詳細資訊，請參閱《AWS CloudFormation 使用者指南》中的[什麼是 CloudFormation 設計器？](#)。

AWS IoT Greengrass 支援在 中建立元件版本和部署 CloudFormation。如需詳細資訊，包括元件版本和部署的 JSON 和 YAML 範本範例，請參閱AWS CloudFormation 《使用者指南》中的[AWS IoT Greengrass 資源類型參考](#)。

ComponentVersion 範本範例

以下是簡單元件版本的 YAML 範本。JSON 配方包含可讀性的換行符號。

```
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 }

```

部署範本範例

以下是定義部署簡單範本的 YAML 檔案。

```

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
```

進一步了解 CloudFormation

若要進一步了解 CloudFormation，請參閱下列資源：

- [AWS CloudFormation](#)
- [AWS CloudFormation 使用者指南](#)
- [CloudFormation API 參考](#)
- [AWS CloudFormation 命令列介面使用者指南](#)

開放原始碼 AWS IoT Greengrass 核心軟體

Core AWS IoT Greengrass 軟體的 AWS IoT Greengrass Version 2 邊緣執行時間（核）和其他元件是開放原始碼。這表示您可以檢閱程式碼來疑難排解與應用程式的互動。您也可以自訂和擴展 AWS IoT Greengrass 核心軟體，以滿足特定的軟體和硬體需求。

如需 AWS IoT Greengrass Core 軟體的開放原始碼儲存庫相關資訊，請參閱 GitHub 上的 [aws-greengrass](#) 組織。您對開放原始碼軟體的使用受[對應 GitHub 儲存庫](#)中的開放原始碼授權所規範。

您對不受開放原始碼授權約束之 AWS IoT Greengrass Core 軟體和元件的使用受 [AWS Greengrass Core 軟體授權](#)的約束。

AWS IoT Greengrass V2 開發人員指南的文件歷史記錄

下表說明此版本 的文件 AWS IoT Greengrass Version 2。

- API 版本：2020-11-30

變更	描述	日期
Shadow Manager v2.3.13 已發行	Shadow Manager v2.3.13 已推出。此版本修正了即使同步方向組態允許，本機陰影更新也會間歇性地無法同步至雲端的問題。	2026 年 2 月 18 日
Greengrass nucleus lite 元件 2.3.2 版已發行	Greengrass nucleus lite 元件的 2.3.2 版已推出。此版本會更新版本檔案，以取得正確的 nucleus 版本報告。	2026 年 2 月 12 日
Greengrass nucleus lite 元件 2.3.1 版已發行	Greengrass nucleus lite 元件的 2.3.1 版已推出。此版本包含數個錯誤修正和改進。	2026 年 2 月 11 日
Greengrass CLI 2.16.1 版已發行	Greengrass CLI 元件 v2.16.1 已推出。	2025 年 12 月 23 日
AWS IoT Greengrass Core v2.16.1 軟體更新	此版本提供 Greengrass 核元件的 2.16.1 AWS 版，以及更新提供的元件。	2025 年 12 月 23 日
系統日誌轉送器 v2.1.0 已發行	系統日誌轉寄站 v2.1.0 已推出。	2025 年 11 月 6 日
Shadow Manager v2.3.12 已發行	Shadow Manager v2.3.12 已推出。	2025 年 11 月 6 日
Log Manager v2.3.11 已發行	Log Manager 元件 v2.3.11 可供使用。	2025 年 11 月 6 日

Greengrass nucleus lite 元件 2.3.0 版已發行	Greengrass nucleus lite 元件的 2.3.0 版已推出。此版本新增了對 TPM 2.0、更廣泛的作業系統和 RestartComponent IPC 的支援。此版本也包含數個其他錯誤修正和改善。	2025 年 11 月 6 日
Greengrass CLI 2.16.0 版已發行	Greengrass CLI 元件 v2.16.0 已推出。	2025 年 11 月 6 日
AWS IoT Greengrass Core v2.16.0 軟體更新	此版本提供 2.16.0 版的 Greengrass 核元件和更新 AWS 提供的元件。	2025 年 11 月 6 日
AWS IoT Greengrass Core v2.15.1 軟體更新	此版本提供 Greengrass 核元件的 2.15.1 AWS 版，以及更新提供的元件。	2025 年 11 月 6 日
系統日誌轉送器 v2.0.1 已發行	系統日誌轉寄站 v2.0.1 可供使用。此版本會更新元件配方，以正確支援 aarch64 (arm64) 系統。	2025 年 10 月 2 日
Greengrass nucleus lite 元件 2.2.2 版已發行	Greengrass nucleus lite 元件的 2.2.2 版已推出。此版本包含數個錯誤修正和改進。	2025 年 8 月 18 日
系統日誌轉送器 v2.0.0 已發行	系統日誌轉送器元件 v2.0.0 可供使用。初始版本已發行。	2025 年 7 月 25 日
Greengrass nucleus lite 元件 2.2.1 版已發行	Greengrass nucleus lite 元件的 2.2.1 版已推出。此版本修正了核心無法取得 TES 登入資料的問題。	2025 年 7 月 25 日

Greengrass nucleus lite 元件 2.2.0 版已發行	Greengrass nucleus lite 元件的 2.2.0 版已推出。此版本新增對容器映像成品 URIs 的支援。	2025 年 7 月 3 日
AWS IoT Greengrass Core v2.15.0 軟體更新	此版本提供 2.15.0 版的 Greengrass 核元件和更新 AWS 提供的元件。	2025 年 7 月 3 日
已發行安全通道 v1.1.3	提供安全通道 v1.1.3。此版本會將元件調用的基礎 AWS IoT Device Client 從 1.10.0 版升級至 1.10.1 版。此版本也修正了安全通道元件 1.1.2 版中的 GNU C 程式庫 (glibc) 相容性問題。	2025 年 5 月 16 日
Greengrass nucleus lite 元件 2.1.0 版已發行	Greengrass nucleus lite 元件的 2.1.0 版已推出。此版本新增 HTTP 代理支援。此版本也包含數個其他錯誤修正和改善。	2025 年 5 月 5 日
Secret Manager 2.2.5 版已發行	Secret Manager 2.2.5 版已推出。此版本修正以下問題： AWS 雲端 如果本機快取中不存在秘密，則不會從中擷取秘密。	2025 年 4 月 16 日
AWS IoT Greengrass Core v2.14.3 軟體更新	此版本提供 Greengrass 核元件的 2.14.3 AWS 版，以及更新提供的元件。	2025 年 4 月 11 日
Secret Manager 2.2.4 版已發行	Secret Manager 2.2.4 版已推出。此版本可減少寫入本機秘密存放區的頻率。Secret Manager 現在只會在秘密更新時寫入本機存放區。	2025 年 4 月 8 日

[Greengrass CLI 2.14.3 版已發行](#)

Greengrass CLI 元件 v2.14.3 已推出。

2025 年 4 月 8 日

[已發行安全通道 v1.1.2](#)

提供安全通道 v1.1.2。此版本會將元件調用的基礎 AWS IoT Device Client 從 1.9.0 版升級至 1.10.0 版。安全通道 v1.1.2 也修正承載傳輸問題，以防止使用者透過安全通道將大型檔案從 Greengrass V2 核心裝置轉送至來源裝置。

2025 年 3 月 27 日

[Greengrass CLI 2.14.2 版已發行](#)

Greengrass CLI 元件 v2.14.2 已推出。

2025 年 3 月 27 日

[AWS IoT Greengrass Core v2.14.2 軟體更新](#)

此版本提供 Greengrass 核元件的 2.14.2 AWS 版，以及更新提供的元件。

2025 年 3 月 27 日

[Secret Manager 2.2.3 版已發行](#)

Secret Manager 2.2.3 版已推出。此版本修正以下問題：當核心裝置離線且裝置的安全服務（例如 HSM）無法使用時，秘密管理員會抹除本機保留的秘密。

2025 年 3 月 18 日

[已發行串流管理員 v2.2.1](#)

串流管理員 v2.2.1 現已推出。此版本修正了串流管理員無法將訊息匯出至 Kinesis 目的地的問題。此版本也會改善串流管理員匯出至 Kinesis 目的地的效能。

2025 年 3 月 12 日

[用戶端裝置驗證元件 v2.5.3 已發行](#)

用戶端裝置驗證元件 v2.5.3 已推出。此版本修正用戶端裝置因用戶端憑證過期而無法連線至核心裝置的問題。

2025 年 3 月 12 日

AWS IoT 已發行機群佈建外掛程式 v1.2.2	AWS IoT 機群佈建外掛程式 v1.2.2 可供使用。此版本新增了對自訂核心裝置憑證 (certificatePath) 和私有金鑰 (privateKeyPath) 路徑的支援。	2025 年 3 月 12 日
已發行安全通道 v1.1.1	提供安全通道 v1.1.1。此版本新增了支援 Greengrass nucleus lite 的組態。	2025 年 2 月 7 日
Greengrass CLI 2.14.1 版已發行	Greengrass CLI 元件 v2.14.1 已推出。	2025 年 2 月 7 日
AWS IoT Greengrass Core v2.14.1 軟體更新	此版本提供 Greengrass 核元件的 2.14.1 AWS 版，以及更新提供的元件。	2025 年 2 月 7 日
Greengrass nucleus lite 元件 2.0.2 版已發行	Greengrass nucleus lite 元件的 2.0.2 版已推出。此版本包含一般錯誤修正和改善。	2025 年 2 月 6 日
Greengrass nucleus lite 元件 2.0.1 版已發行	Greengrass nucleus lite 元件的 2.0.1 版已推出。此版本包含一般錯誤修正和改善。	2025 年 1 月 28 日
Greengrass CLI 2.14.0 版已發行	Greengrass CLI 元件 v2.14.0 已推出。	2024 年 12 月 24 日
已發行串流管理員 v2.2.0	串流管理員 v2.2.0 現已推出。	2024 年 12 月 16 日
Shadow Manager v2.3.10 已發行	Shadow Manager v2.3.10 已推出。	2024 年 12 月 16 日
已發行安全通道 v1.1.0	提供安全通道 v1.1.0。此版本新增 Greengrass nucleus lite 的配方支援。	2024 年 12 月 16 日

新的核遙測發射器元件	1.0.10 版的晶核遙測發射器元件已推出。	2024 年 12 月 16 日
Modbus-RTU 通訊協定轉接器 v2.1.10 發行	Modbus-RTU 通訊協定轉接器元件 v2.1.10 已推出。	2024 年 12 月 16 日
Log Manager 2.3.9 版已發行	Log Manager 元件 v2.3.9 可供使用。	2024 年 12 月 16 日
發行本機偵錯主控台 v2.4.4	本機偵錯主控台元件 v2.4.4 已推出。此版本包含一般錯誤修正和改善。	2024 年 12 月 16 日
Lambda Manager 2.3.5 版已發行	Lambda Manager 元件 v2.3.5 已推出。	2024 年 12 月 16 日
IP 偵測器 2.2.1 版已發行	IP 偵測器元件 v2.2.1 已推出。	2024 年 12 月 16 日
Greengrass nucleus lite 元件 2.0.0 版已發行	Greengrass nucleus lite 元件的 2.0.0 版已推出。這是初始版本。	2024 年 12 月 16 日
磁碟多工緩衝處理器 1.0.5 版已發行	磁碟多工緩衝處理元件 v1.0.5 已推出。	2024 年 12 月 16 日
用戶端裝置驗證元件 v2.5.2 已發行	用戶端裝置驗證元件 v2.5.2 已推出。	2024 年 12 月 16 日

[AWS IoT Greengrass Core](#) [v2.14.0 軟體更新](#)

2024 年 12 月 16 日

此版本提供 Greengrass 核元件的 2.14.0 版，以及新的 AWS IoT Greengrass 核精簡更新。AWS IoT Greengrass nucleus lite 是新的執行時間，適用於 AWS IoT Greengrass 第 2 版。它提供了減少記憶體使用量的替代方案。對於資源受限的裝置來說，這是不錯的選項。它實作了一部分的核功能，具有針對未來版本規劃的更多特色相容性。來源碼現在可在 [Github](#) 上使用。使用 nucleus lite 執行時間，您可以：

- 將元件部署到 Greengrass 核心裝置。使用相同的配方格式，但某些進階功能可能尚無法使用。
- 部署為 Greengrass 元件的應用程式可以使用裝置 SDKs 來存取支援的 Greengrass IPC APIs，例如：AWS IoT Core MQTT 存取、本機 pub/sub 和 Greengrass 組態存取。如需[支援的 IPC APIs](#) 清單，請參閱相容性圖表。
- 部分 AWS 受管元件已更新，支援 nucleus lite。如需現有相容[AWS 元件的清單](#)，請參閱[提供的](#)元件。

新功能：

- 使用較少的記憶體和磁碟空間（少於 5MB 的 RAM 和少於 5MB 的儲存空間）。
- 元件與主機系統的服務管理員整合（適用於目前支援的 Linux 平台）。

要注意的事項：

- AWS IoT Greengrass nucleus lite 配方區分大小寫。確保使用正確的（金鑰）大小寫，如<https://docs.aws.amazon.com/greengrass/v2/developerguide/component-recipe-reference.html>配方參考中所示。
- nucleus lite 執行時間支援物件群組部署，但尚不支援（單一）Core 裝置部署目標類型。若要部署到單一 Greengrass 裝置，請使用物件群組，其中只有一個裝置。
- 核精簡型執行期使用邊界記憶體資源；根據傳統執行期的使用量擴展的功能可能會因為超過精簡型資源而失敗。這包括一次最多 50 個 MQTT 訂閱的目前限制，以及配方檔案大小和部署的最大限制。如果自行編譯精簡型執行時間，則可以在編譯時設定其中一些限制。

- nucleus lite 執行時間不會隨附於 Java。若要使用需要 Java 的元件，系統將需要已安裝 Java，或者可以使用元件來安裝 Java。
- 我們建議您從來源編譯 nucleus lite 執行時間，並使用專為您的系統量身打造的建置。對於 Yocto 系統，layer 可用於將 nucleus lite 執行時間整合到您的系統映像中。
- 目前，nucleus lite 假設使用 systemd 的 Linux 系統，或使用 systemd 的容器映像。
- 雖然您可以使用配方指令碼管理 Docker 容器，但 Greengrass 受管容器成品尚無法使用。
- nucleus lite 執行時間尚未支援存放在 PKCS11 模組中的金鑰。如果您的使用案例需要存放在安全元素上的金鑰，則傳統執行時間目前可支援此使用案例。為了防止裝置登入資料洩漏，請確保生產裝置使用完整磁碟加密。

除了推出 nucleus lite 之外，我們也發行了 nucleus v2.14.0。此更新為現有的 Greengrass 核帶來顯著的增強功能。

主要功能和改進：

- 新的雙堆疊端點支援可啟用 IPv6 網路通訊。
- 增強對核重新啟動失敗和目錄損毀的彈性。
- 修正 IPC PubSub 訂閱關閉中的記憶體流失問題。

[串流管理員 v2.1.13 已發行](#)

串流管理員 v2.1.13 現已推出。此版本新增對的 FIPS 端點的支援 AWS IoT SiteWise

2024 年 8 月 26 日

[Shadow Manager 2.3.9 版已發行](#)

Shadow Manager 2.3.9 版已推出。

2024 年 8 月 26 日

[Modbus-RTU 通訊協定轉接器 2.1.9 版已發行](#)

Modbus-RTU 通訊協定轉接器元件 v2.1.9 已推出。

2024 年 8 月 26 日

[Log Manager 2.3.8 版已發行](#)

Log Manager 元件 v2.3.8 已推出。

2024 年 8 月 26 日

[已發行本機偵錯主控台 v2.4.3](#)

本機偵錯主控台元件 v2.4.3 已推出。此版本包含一般錯誤修正和改善。

2024 年 8 月 26 日

[磁碟多工緩衝處理器 1.0.4 版已發行](#)

磁碟多工緩衝處理元件 v1.0.4 已推出。

2024 年 8 月 26 日

[AWS IoT Greengrass Core v2.13.0 軟體更新](#)

此版本提供 Greengrass 核元件的 2.13.0 版。

2024 年 8 月 26 日

[新的核遙測發射器元件](#)

1.0.9 版的 核遙測發射器元件已推出。

2024 年 8 月 23 日

[Lambda Manager 2.3.4 版已發行](#)

Lambda Manager 元件 v2.3.4 已推出。

2024 年 8 月 23 日

[Greengrass CLI 2.13.0 版已發行](#)

Greengrass CLI 元件 v2.13.0 已推出。

2024 年 8 月 23 日

用戶端裝置驗證元件 v2.5.1 已發行	用戶端裝置驗證元件 v2.5.0 已推出。此版本新增對 FIPS 端點的支援。	2024 年 8 月 23 日
配方驗證	新增配方驗證功能，可在建立元件版本時驗證元件配方。	2024 年 8 月 15 日
IP 偵測器 2.2.0 版已發行	IP 偵測器元件 v2.2.0 已推出。此版本新增對 IPv6 的支援。您現在可以將 IPv6 用於本機傳訊。	2024 年 7 月 29 日
Shadow Manager 2.3.8 版已發行	Shadow Manager 2.3.8 版已推出。此版本修正了陰影管理員在 MQTT 用戶端連線期間建立死結情況的問題。	2024 年 6 月 5 日
Greengrass CLI 2.12.6 版已發行	Greengrass CLI 元件 v2.12.6 已推出。	2024 年 5 月 24 日
AWS IoT Greengrass Core v2.12.6 軟體更新	此版本提供 2.12.6 版的 Greengrass 核元件和更新 AWS 提供的元件。	2024 年 5 月 24 日
AWS IoT Device Tester v4.9.4 搭配發行的 GGV2Q v2.5.4	IDT for AWS IoT Greengrass V2 4.9.4 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.5.4 版，並支援 Greengrass 核版本 2.12.0、2.11.0、2.10.0、2.9.5。	2024 年 5 月 3 日

已發行安全通道 v1.0.19	提供安全通道 v1.0.19。此版本會將元件調用的基礎 AWS IoT Device Client 從 1.8.0 版升級至 1.9.0 版。安全通道 v1.0.19 會將同時通道限制提高到元件層級的 20 個通道。這個新版本也會將 AWS IoT Greengrass 核心 IPC 逾時從 3 秒增加到 10 秒。	2024 年 5 月 1 日
Kinesis Video Streams 元件 1.0.5 版的邊緣連接器已發行	Kinesis Video Streams 元件的邊緣連接器 1.0.5 版已推出。此版本包含一般錯誤修正和改善。	2024 年 4 月 29 日
Greengrass CLI 2.12.5 版已發行	Greengrass CLI 元件 v2.12.5 已推出。	2024 年 4 月 25 日
用戶端裝置驗證元件 v2.5.0 已發行	用戶端裝置驗證元件 v2.5.0 已推出。此版本新增物件名稱的政策變數支援。此版本也允許使用萬用字元的政策資源。	2024 年 4 月 25 日
AWS IoT Greengrass Core v2.12.5 軟體更新	此版本提供 Greengrass 核元件的 2.12.5 AWS 版，以及更新提供的元件。	2024 年 4 月 25 日
AWS IoT Device Tester v4.9.3 搭配發行的 GGV2Q v2.5.3	IDT for AWS IoT Greengrass V2 4.9.3 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.5.3 版，並支援 Greengrass 核版本 2.12.0、2.11.0、2.10.0、2.9.5。	2024 年 4 月 5 日
Greengrass CLI 2.12.4 版已發行	Greengrass CLI 元件 v2.12.4 已推出。	2024 年 4 月 2 日

AWS IoT Greengrass Core v2.12.4 軟體更新	此版本提供 Greengrass 核元件的 2.12.4 AWS版，以及更新提供的元件。	2024 年 4 月 2 日
Shadow Manager 2.3.7 版已發行	Shadow Manager 2.3.7 版已推出。此版本修正影子管理員在影子管理員同步期間定期記錄NullPointerException 錯誤的問題。	2024 年 3 月 27 日
Moquette MQTT 3.1.1 代理程式 2.3.6 版已發行	Moquette MQTT 3.1.1 代理程式元件 v2.3.6 已推出。此版本包含一般錯誤修正和改善。	2024 年 3 月 27 日
發行本機偵錯主控台 v2.4.2	本機偵錯主控台元件 v2.4.2 可供使用。此版本包含一般錯誤修正和改善。	2024 年 3 月 27 日
Lambda Manager 2.3.3 版已發行	Lambda Manager 元件 v2.3.3 已推出。此版本包含一般錯誤修正和改善。	2024 年 3 月 27 日
IP 偵測器 2.1.9 版已發行	IP 偵測器元件 v2.1.9 已推出。此版本會調整 IP 取得的步驟，以僅在偵錯日誌層級傳送日誌。	2024 年 3 月 27 日
AWS IoT 已發行機群佈建外掛程式 v1.2.1	AWS IoT 機群佈建外掛程式 v1.2.1 可供使用。此版本修正了 Greengrass 核啟動期間機群佈建外掛程式離線的問題。機群佈建外掛程式現在會無限期重試 MQTT 連線呼叫。	2024 年 3 月 27 日
AWS IoT Greengrass Core v2.12.3 軟體更新	此版本提供 Greengrass 核元件的 2.12.3 AWS版，以及更新提供的元件。	2024 年 3 月 27 日

Greengrass CLI 2.12.3 版已發行	Greengrass CLI 元件 v2.12.3 已推出。	2024 年 3 月 25 日
AWS IoT Device Tester v4.9.2 搭配發行的 GGV2Q v2.5.2	IDT AWS IoT Greengrass forV2 4.9.2 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.5.2 版，並支援 Greengrass 核版本 2.12.0、2.11.0、2.10.0、2.9.5。	2024 年 3 月 18 日
AWS IoT Greengrass Core v2.12.2 軟體更新	此版本提供 Greengrass 核元件的 2.12.2 AWS 版，以及更新提供的元件。	2024 年 2 月 15 日
Shadow Manager 2.3.6 版已發行	Shadow Manager 2.3.6 版已推出。此版本修正以下問題：重新取得連線後，裝置離線時透過 AWS 雲端 更新刪除的影子屬性會繼續存在於本機影子中。	2024 年 2 月 14 日
Lambda 啟動器 2.0.13 版已發行	Lambda 啟動器元件的 2.0.13 版已推出。此版本包含一般錯誤修正和改善。	2024 年 2 月 14 日
磁碟多工緩衝處理器 1.0.3 版已發行	磁碟多工緩衝處理元件 v1.0.3 已推出。此版本透過重複使用資料庫連線來改善效能。	2024 年 2 月 14 日
Greengrass 開發套件 CLI 1.6.2 版	Greengrass 開發套件 CLI 的 1.6.2 版已推出。此版本修正 Windows gradlew.bat 因相對路徑而無法運作的問題。此版本也包含其他改善項目。	2024 年 1 月 16 日

新的 CloudTrail 資料事件	您現在可以記錄 AWS CloudTrail 資料事件，以取得資源操作的相關資訊，例如取得元件或部署的組態。使用這些事件來深入了解 Greengrass 裝置的操作。	2023 年 12 月 20 日
串流管理員 v2.1.12 已發行	串流管理員 v2.1.12 現已推出。此版本會變更 Greengrass 用來選取一組 AWS 服務呼叫登入資料的順序。	2023 年 12 月 8 日
MQTT 橋接器 2.3.1 版已發行	MQTT 橋接器 v2.3.1 已推出。此版本修正了本機 MQTT 用戶端進入中斷連線迴圈的罕見問題。	2023 年 12 月 8 日
磁碟多工緩衝處理器 1.0.2 版已發行	磁碟多工緩衝處理器元件 v1.0.2 已推出。此版本修正在某些情況下未保留 MQTT 訊息格式欄位的問題。	2023 年 12 月 8 日
用戶端裝置驗證元件 v2.4.5 已發行	用戶端裝置驗證元件 v2.4.5 已推出。此版本在選取規則的物件名稱結尾新增對萬用字元的支援，並修正在特定情況下，憑證未以新的連線資訊更新的問題。	2023 年 12 月 8 日
AWS IoT Greengrass Core v2.12.1 軟體更新	此版本提供 Greengrass 核心元件的 2.12.1 AWS 版，以及更新提供的元件。	2023 年 12 月 8 日
Greengrass 開發套件 CLI 1.6.1 版	Greengrass 開發套件 CLI 的 1.6.1 版已推出。此版本包含錯誤修正和改善。	2023 年 12 月 6 日

配方驗證	新增配方驗證功能，可在建立元件版本時驗證元件配方。	2023 年 11 月 16 日
發行者支援的元件	AWS IoT Greengrass 現在提供發佈者支援的元件。這些元件是由第三方供應商開發、提供和服務。	2023 年 11 月 16 日
Greengrass 測試架構 1.2.0 版已發行	Greengrass 測試架構 1.2.0 版已推出。	2023 年 11 月 15 日
Greengrass 開發套件 CLI 1.6.0 版	Greengrass 開發套件 CLI 的 1.6.0 版已推出。此版本會在 component build 和 component publish 命令期間，針對 Greengrass 配方結構描述新增配方驗證檢查。此更新有助於開發人員在元件建立程序中稍早在其元件配方中識別可採取動作的問題。此版本也會將可信度測試套件新增至範本，該範本可由 test-e2e init 命令提取。此可信度測試套件包含八項通用測試，可用於和擴展以滿足基本元件測試需求。	2023 年 11 月 15 日
AWS IoT Device Tester v4.9.1 支援 Greengrass 核 2.12.0 版	IDT for AWS IoT Greengrass V2 4.9.1 版現在支援 Greengrass 核 2.12.0 版。	2023 年 11 月 7 日
AWS IoT Greengrass Core v2.12.0 軟體更新	此版本提供 Greengrass 核元件的 2.12.0 AWS 版，以及更新提供的元件。	2023 年 11 月 7 日

在 VPC 中操作 Greengrass 核心裝置	可在 VPC 中操作 Greengrass 核心裝置。此功能可讓您在沒有公有網際網路存取的情況下，在 VPC 中執行部署。	2023 年 11 月 3 日
Greengrass CLI 2.12.0 版已發行	Greengrass CLI 元件 v2.12.0 已推出。	2023 年 10 月 30 日
串流管理員 v2.1.10 已發行	串流管理員 v2.1.10 現已推出。此版本修正 HTTPS 代理組態不信任 Greengrass CA 憑證鏈的問題。	2023 年 10 月 26 日
Lambda 啟動器 2.0.12 版已發行	Lambda 啟動器元件的 2.0.12 版已推出。此版本修正以下問題：如果先前的程序未正確停止，Lambda 啟動器可能會擲回錯誤。	2023 年 10 月 26 日
Greengrass 開發套件 CLI 1.5.0 版	Greengrass 開發套件 CLI 的 1.5.0 版已推出。當 build_system 為時，此版本會更新excludes建置選項所辨識的模式zip。此版本現在會根據萬用字元來辨識符合路徑名稱的 glob 模式。這會啟用要排除哪些目錄的自訂規格。	2023 年 10 月 26 日
Shadow Manager 2.3.4 版已發行	Shadow Manager 2.3.4 版已推出。此版本新增對 null 和空白陰影狀態文件的支援。	2023 年 10 月 18 日
Log Manager 2.3.6 版已發行	Log Manager 元件 v2.3.6 已推出。	2023 年 10 月 18 日
本機偵錯主控台 2.4.0 版已發行	本機偵錯主控台元件 v2.4.0 已推出。	2023 年 10 月 18 日

Lambda Manager 2.3.1 版已發行	Lambda Manager 元件 v2.3.1 已推出。	2023 年 10 月 18 日
Greengrass CLI 2.11.3 版已發行	Greengrass CLI 元件 v2.11.3 已推出。	2023 年 10 月 18 日
AWS IoT Greengrass Core v2.11.3 軟體更新	此版本提供 Greengrass 核元件的 2.11.3 AWS 版，以及更新提供的元件。	2023 年 10 月 18 日
已發行安全通道 v1.0.17	提供安全通道 v1.0.17。	2023 年 10 月 4 日
Greengrass 開發套件 CLI 1.4.0 版	Greengrass 開發套件 CLI 的 1.4.0 版已推出。此版本新增了新的 config 命令，會啟動互動式提示來修改現有 GDK 組態檔案中的欄位。此版本也會修改 gdk component build 和 gdk component publish 命令，以確認配方法大小在 Greengrass 要求 (≤ 16000 位元組) 內，然後再繼續。	2023 年 10 月 2 日
Moquette MQTT 3.1.1 代理程式 2.3.5 版已發行	Moquette MQTT 3.1.1 代理程式元件 v2.3.5 已推出。此版本將 Moquette 更新至 0.17 版。	2023 年 9 月 28 日
MQTT 橋接器 v2.3.0 已發行	MQTT 橋接器 v2.3.0 已推出。此版本新增 MQTT 5 支援橋接 AWS IoT Core 和本機 MQTT 來源。	2023 年 9 月 28 日
Lambda Manager 2.3.0 版已發行	Lambda Manager 元件 v2.3.0 已推出。	2023 年 9 月 15 日

Lambda 啟動器 2.0.11 版已發行	Lambda 啟動器元件的 2.0.11 版已推出。此版本支援 Lambda Manager 2.3.0。	2023 年 9 月 15 日
Moquette MQTT 3.1.1 代理程式 2.3.4 版已發行	Moquette MQTT 3.1.1 代理程式元件 v2.3.4 已推出。	2023 年 9 月 1 日
Greengrass 測試架構	GTF 是支援end-to-end自動化的建置區塊集合。它可讓 AWS IoT Greengrass Version 2 內部客戶使用與服務團隊用於合格軟體變更、自動接受和品質保證目的相同的測試架構。	2023 年 8 月 11 日
AWS IoT Greengrass Core v2.11.2 軟體更新	此版本提供 Greengrass 核元件的 2.11.2 AWS版，以及更新提供的元件。	2023 年 8 月 9 日
Greengrass 開發套件 CLI 1.3.0 版	Greengrass 開發套件 CLI 的 1.3.0 版已推出。此版本新增了新的test-e2e命令，以支援使用 Open Test Framework 進行元件的end-to-end測試。	2023 年 7 月 21 日
AWS IoT Greengrass Core v2.11.1 軟體更新	此版本提供 Greengrass 核元件的 2.11.1 AWS版，以及更新提供的元件。	2023 年 7 月 21 日
磁碟多工緩衝處理器 1.0.0 版已發行	磁碟多工緩衝處理元件 v1.0.0 已推出。	2023 年 6 月 28 日
AWS IoT Greengrass Core v2.11.0 軟體更新	此版本提供 Greengrass 核元件的 2.11.0 AWS版，以及更新提供的元件。	2023 年 6 月 28 日
AWS IoT Greengrass Core v2.10.3 軟體更新	此版本提供 Greengrass 核元件的 2.10.3 AWS版，以及更新提供的元件。	2023 年 6 月 21 日

AWS IoT Greengrass Core v2.10.2 軟體更新	此版本提供 Greengrass 核元件的 2.10.2 AWS 版，以及更新提供的元件。	2023 年 6 月 5 日
AWS IoT Greengrass Core v2.10.1 軟體更新	此版本提供 Greengrass 核元件的 2.10.1 AWS 版，以及更新提供的元件。	2023 年 5 月 11 日
AWS IoT Greengrass Core v2.10.0 軟體更新	此版本提供 2.10.0 版的 Greengrass 核元件和更新 AWS 提供的元件。	2023 年 5 月 9 日
SageMaker AI Edge Manager 已停止	Amazon SageMaker AI Edge Manager 元件已於 2024 年 4 月 26 日停用。	2023 年 4 月 28 日
AWS IoT Greengrass Core v2.9.6 軟體更新	此版本提供 2.9.6 版的 Greengrass 核元件和更新 AWS 提供的元件。	2023 年 4 月 20 日
Log Manager 2.3.2 版已發行	Log Manager 元件 v2.3.2 已推出。	2023 年 4 月 19 日
串流管理員 2.1.4 版已發行	串流管理員 v2.1.4 現已推出。此版本修正以下問題： 在 SiteWise API Conflicti ngOperationException on 的單一批次傳回中，具有 相同時間戳記的相同屬性資產 項目導致串流管理員持續重 試。此版本也會將預設連線逾 時從 3 秒更新為 1 分鐘。	2023 年 4 月 13 日
Greengrass 開發套件 CLI 1.2.3 版	Greengrass 開發套件 CLI 的 1.2.3 版已推出。此版本包含錯誤修正。	2023 年 4 月 13 日

用戶端裝置驗證元件 v2.4.0 已發行	用戶端裝置驗證元件 v2.4.0 已推出。此版本新增對用戶端裝置身分驗證的支援，以發出可在 Greengrass 用戶端裝置儀表板上顯示的操作指標。	2023 年 4 月 10 日
Greengrass 開發套件 CLI 1.2.2 版	Greengrass 開發套件 CLI 的 1.2.2 版已推出。此版本包含改善和錯誤修正。	2023 年 4 月 7 日
AWS IoT Greengrass Core v2.9.5 軟體更新	此版本提供 Greengrass 核元件的 2.9.5 AWS 版，以及更新提供的元件。	2023 年 3 月 30 日
已發行串流管理員 v2.1.3	串流管理員 v2.1.3 現已推出。此版本修正以 SYSTEM 使用者身分執行時 Windows 作業系統上的啟動問題。	2023 年 3 月 7 日
Modbus-RTU 通訊協定轉接器 2.1.5 版已發行	Modbus-RTU 通訊協定轉接器元件 v2.1.5 已推出。此版本修正了 ReadDiscreteInput 操作的問題。	2023 年 3 月 7 日
用戶端裝置驗證元件 v2.3.2 已發行	用戶端裝置驗證元件 v2.3.2 已推出。此版本新增快取主機名稱資訊的支援，以便元件在離線重新啟動時正確產生憑證主體。	2023 年 3 月 7 日
AWS IoT Device Tester v4.7.0 支援 Greengrass 核 2.9.4 版	IDT for AWS IoT Greengrass V2 4.7.0 版現在支援 Greengrass 核 2.9.4 版。	2023 年 3 月 2 日
Greengrass 命令列介面 1.2.0 版已發行	Greengrass 命令列介面 1.2.0 版已推出。	2023 年 2 月 28 日

AWS IoT Greengrass Core v2.9.4 軟體更新	此版本提供 Greengrass 核元件的 2.9.4 AWS 版，以及更新提供的元件。	2023 年 2 月 24 日
Shadow Manager 2.3.1 版已發行	Shadow Manager 2.3.1 版已推出。此版本修正了可能導致雲端影子更新無法同步的條件。此版本也修正了對具名影子同步組態的變更僅適用於一個具名影子的問題。	2023 年 2 月 21 日
AWS IoT Device Tester v4.7.0 支援 Greengrass 核 2.9.3 版	IDT for AWS IoT Greengrass V2 4.7.0 版現在支援 Greengrass 核 2.9.3 版。	2023 年 2 月 9 日
IAM 最佳實務已更新	更新了指南以符合 IAM 最佳實務。如需更多詳細資訊，請參閱 IAM 中的安全最佳實務 。	2023 年 2 月 3 日
AWS IoT Greengrass Core v2.9.3 軟體更新	此版本提供 Greengrass 核元件的 2.9.3 AWS 版，以及更新提供的元件。	2023 年 2 月 1 日
Log Manager 2.3.1 版已發行	Log Manager 2.3.1 版已推出。	2023 年 1 月 27 日
AWS IoT Device Tester v4.7.0 支援 Greengrass 核 2.9.2 版	IDT for AWS IoT Greengrass V2 4.7.0 版現在支援 Greengrass 核 2.9.2 版。	2023 年 1 月 3 日
Shadow Manager 2.3.0 版已發行	Shadow Manager 2.3.0 版已推出。此版本修正了當裝置將 Greengrass 裝置私有金鑰存放在硬體安全模組中時，可能阻止陰影同步的問題。	2022 年 12 月 29 日

AWS IoT 已發行機群佈建外掛程式 v1.2.0	AWS IoT 機群佈建外掛程式 v1.2.0 可供使用。此版本透過可設定私有金鑰路徑的憑證簽署請求新增裝置佈建的支援。	2022 年 12 月 22 日
AWS IoT Greengrass Core v2.9.2 軟體更新	此版本提供 Greengrass 核元件的 2.9.2 AWS 版，以及更新提供的元件。	2022 年 12 月 22 日
AWS IoT Device Tester v4.7.0 搭配發行的 GGV2Q v2.5.0	IDT for AWS IoT Greengrass V2 4.7.0 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.5.0 版，並支援 Greengrass 核版本 2.9.1、2.9.0、2.8.1、2.8.0、2.7.0 和 2.6.0。	2022 年 12 月 13 日
Shadow Manager 2.2.4 版已發行	修正更新本機影子文件時，影子大小驗證與雲端不一致的問題。如果部署在組態節點 RESET 上執行，這也修正了陰影管理員停止接聽組態更新的問題。	2022 年 12 月 8 日
Log Manager 2.3.0 版已發行	Log Manager 元件 v2.3.0 已推出。	2022 年 11 月 18 日
AWS IoT Device Tester v4.5.11 支援 Greengrass nucleus 2.9.1 版	IDT for AWS IoT Greengrass V2 的 4.5.11 版現在支援 Greengrass 核 2.9.1 版。	2022 年 11 月 18 日
AWS IoT Greengrass Core v2.9.1 軟體更新	此版本提供 Greengrass 核元件的 2.9.1 AWS 版，以及更新提供的元件。	2022 年 11 月 18 日
AWS IoT Device Tester v4.5.11 支援 Greengrass nucleus 2.9.0 版	IDT for AWS IoT Greengrass V2 的 4.5.11 版現在支援 Greengrass 核 2.9.0 版。	2022 年 11 月 17 日

已發行串流管理員 v2.1.2	串流管理員 v2.1.2 現已推出。此版本修正了使用非英文語言的 Windows 作業系統問題。	2022 年 11 月 15 日
AWS IoT Greengrass Core v2.9.0 軟體更新	此版本提供 Greengrass 核元件的 2.9.0 AWS 版，以及更新提供的元件。	2022 年 11 月 15 日
AWS IoT Device Tester v4.5.11 支援 Greengrass 核版本 2.8.1	IDT for AWS IoT Greengrass V2 的 4.5.11 版現在支援 Greengrass 核 2.8.1 版。	2022 年 10 月 19 日
AWS IoT Device Tester v4.5.11，已發行 GGV2Q v2.4.1	IDT for AWS IoT Greengrass V2 的 4.5.11 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.4.1 版，並支援 Greengrass 核版本 2.8.0、2.7.0 和 2.6.0。	2022 年 10 月 13 日
AWS IoT Greengrass Core v2.8.1 軟體更新	此版本提供 Greengrass 核元件的 2.8.1 AWS 版，以及更新提供的元件。	2022 年 10 月 13 日
AWS IoT Greengrass Core v2.8.0 軟體更新	此版本提供 Greengrass 核元件的 2.8.0 AWS 版，以及更新提供的元件。	2022 年 10 月 7 日
新增對部署的 CloudFormation 支援	CloudFormation 現在支援 AWS IoT Greengrass 部署做為資源。	2022 年 10 月 6 日
SageMaker AI Edge Manager 1.3.0 版已發行	Amazon SageMaker AI Edge Manager 元件 1.3.0 版已推出。此版本新增了此元件的支援，以設定 TensorRT 模型快取的磁碟大小，並改善預測並行，以更好地利用 GPUs 等裝置加速器引擎。	2022 年 9 月 1 日

[使用程序間通訊 \(IPC\) 用戶端 V2](#)

新增有關 IPC 用戶端 V2 的資訊，這可減少您需要寫入以使用 IPC 操作的程式碼量，並有助於避免 IPC 用戶端 V1 可能發生的常見錯誤。

2022 年 8 月 12 日

[AWS IoT Device Tester v4.5.8 搭配發行的 GGV2Q v2.4.0](#)

IDT for AWS IoT Greengrass V2 4.5.8 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.4.0 版，並支援 Greengrass 核版本 2.7.0、2.6.0 和 2.5.6。

2022 年 8 月 12 日

[SageMaker AI Edge Manager 1.2.0 版已發行](#)

Amazon SageMaker AI Edge Manager 元件 1.2.0 版已推出。此版本新增此元件的支援，可自動擷取您上傳至 Amazon S3 的 SageMaker AI Neo 編譯模型，因此您可以部署新的模型，而無需建立 AWS IoT Greengrass 部署。

2022 年 8 月 3 日

[AWS IoT Device Tester v4.5.3 支援 Greengrass 核 2.7.0 版](#)

IDT for AWS IoT Greengrass V2 4.5.3 版現在支援 Greengrass 核 2.7.0 版。

2022 年 8 月 1 日

[已發行串流管理員 v2.1.0](#)

串流管理員 v2.1.0 現已推出。此版本包含支援您將遙測指標傳送至 Amazon EventBridge。

2022 年 7 月 28 日

[AWS IoT Greengrass Core v2.7.0 軟體更新](#)

此版本提供 Greengrass 核元件的 2.7.0 AWS 版，以及更新提供的元件。它包含支援您將遙測指標傳送至 Amazon EventBridge。

2022 年 7 月 28 日

[IoT SiteWise 發佈者 2.2.0 版已發行](#)

IoT SiteWise 發佈者元件 v2.2.0 已推出。此版本會更新元件，在將資料傳送到 AWS IoT SiteWise 服務之前壓縮資料，從而減少高達 75% 的頻寬使用量。

2022 年 7 月 19 日

[教學課程：開發與用戶端裝置陰影互動的元件](#)

已將新模組新增至[教學課程：透過 MQTT 與本機 IoT 裝置互動](#)，您可以遵循這些 MQTT 來了解如何開發與用戶端裝置影子互動的元件。

2022 年 7 月 18 日

[選擇本機 MQTT 代理程式](#)

新增如何選擇用戶端裝置連接到核心裝置之本機 MQTT 代理程式的相關資訊。

2022 年 7 月 18 日

[AWS IoT Device Tester v4.5.3 支援 Greengrass 核 2.6.0 版](#)

IDT for AWS IoT Greengrass V2 4.5.3 版現在支援 Greengrass 核 2.6.0 版。

2022 年 6 月 29 日

[AWS IoT Greengrass Core v2.6.0 軟體更新](#)

此版本提供 Greengrass 核元件的 2.6.0 AWS 版，以及更新提供的元件。它包含對用戶端裝置陰影的支援，以及用戶端裝置的本機 MQTT 5 代理程式。它還包含支援本機發佈/訂閱主題中的萬用字元、元件組態中的配方變數，以及 IPC 授權政策中的萬用字元。這些功能可讓您更輕鬆地開發和設定部署到核心裝置機群的元件。此版本也包含支援元件使用 IPC 操作來管理本機部署和核心裝置上的元件。

2022 年 6 月 27 日

用戶端裝置元件更新	用戶端裝置驗證 v2.1.0、 MQTT 代理程式 (Moquette) v2.1.0、 MQTT 橋接器 v2.1.1 和 IP 偵測器 v2.1.2 可供使用。此版本可改善憑證輪換、改善 MQTT 代理程式效能，並修正這些元件如何處理組態重設更新的問題。	2022 年 6 月 14 日
AWS IoT Device Tester v4.5.3 支援 Greengrass 核 2.5.6 版	IDT for AWS IoT Greengrass V2 4.5.3 版現在支援 Greengrass 核 2.5.6 版。	2022 年 6 月 1 日
AWS IoT Greengrass Core v2.5.6 軟體更新	此版本提供 2.5.6 版的 Greengrass 核元件和更新 AWS 提供的元件。它包含對具有 ECC 金鑰的硬體安全模組的支援。它還包含其他錯誤修正和改進。	2022 年 5 月 31 日
AWS IoT 已發行機群佈建外掛程式 v1.1.0	AWS IoT 機群佈建外掛程式 v1.1.0 可供使用。當您在 Windows 裝置上設定外掛程式時，此版本新增對其他檔案路徑格式的支援。	2022 年 5 月 12 日
已發佈新的 Lambda 執行時間	新增對新的 Lambda 執行時間的支援：Python 3.9、Java 11 和 NodeJS 14。	2022 年 5 月 10 日
開發可延遲元件更新的 Greengrass 元件	新增了教學課程，您可以遵循此教學課程來了解如何開發 Greengrass 元件，以延遲部署的元件更新。例如，當裝置電池電量低或執行無法中斷的程序時，您可能想要延遲更新。	2022 年 5 月 4 日

[CloudWatch 指標 v3.1.0 和 AWS IoT Device Defender v3.1.0 已發行](#)

CloudWatch 指標元件 v3.1.0 和 AWS IoT Device Defender 元件 v3.1.0 已推出。這些版本新增了對 HTTPS 網路代理組態的支援。如需詳細資訊，請參閱在[連接埠 443 上或透過網路代理連線](#)，以及[啟用核心裝置來信任 HTTPS 代理](#)。

2022 年 4 月 27 日

[從 遷移 AWS IoT Greengrass Version 1](#)

新增了指南，供您遵循以從 遷移 AWS IoT Greengrass V1 至 AWS IoT Greengrass V2。

2022 年 4 月 26 日

[AWS IoT Device Tester 已更新 v4.5.3 搭配 GGV2Q v2.3.1，並將 IDT v4.5.1 搭配 GGV2Q v2.3.0 新增至支援的版本](#)

IDT for AWS IoT Greengrass V2 with AWS IoT Greengrass V2 資格套件 (GGV2Q) 版本 4.5.3 已更新為包含對 Greengrass 核版本 2.5.5、2.5.4 和 2.5.3 的支援。此更新也包含 IDT 4.5.1 搭配 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.3.0 版做為支援的版本。IDT 4.5.1 with AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.3.0 版支援 Greengrass 核 2.5.3 版。

2022 年 4 月 25 日

[Modbus-RTU 通訊協定轉接器 v2.1.0 已發行](#)

Modbus-RTU 通訊協定轉接器元件 v2.1.0 已推出。此版本新增了新參數，您可以指定這些參數來設定與 Modbus RTU 裝置的序列通訊。

2022 年 4 月 20 日

[發行 CloudWatch 指標
v2.1.0、Firehose v2.1.0 和
Amazon SNS v2.1.0](#)

CloudWatch 指標元件
v2.1.0、Firehose 元件 v2.1.0
和 Amazon SNS 元件 v2.1.0
已推出。這些版本新增了對
HTTPS 網路代理組態的支援
。如需詳細資訊，請參閱在[連接埠 443 上或透過網路代理連線](#)，以及[啟用核心裝置來信任 HTTPS 代理](#)。

2022 年 4 月 19 日

[AWS IoT Device Tester v4.5.3
搭配發行的 GGV2Q v2.3.1](#)

IDT for AWS IoT Greengrass
V2 4.5.3 版已推出。此版本包
含 AWS IoT Greengrass V2 資
格套件 (GGV2Q) 2.3.1 版，並
支援 Greengrass 核 2.5.5 版。

2022 年 4 月 15 日

[AWS IoT Greengrass Core
v2.5.5 軟體更新](#)

此版本提供 Greengrass 核元
件的 2.5.5 AWS 版，以及更新
提供的元件。它新增了對使用
英文以外顯示語言之 Windows
裝置的支援。它還修正了核心
裝置在特定情況下佈建後未向
AWS IoT Greengrass 雲端服
務報告其狀態的問題。

2022 年 4 月 6 日

[AWS IoT Greengrass Core
v2.5.4 軟體更新](#)

此版本提供 Greengrass 核元
件的 2.5.4 AWS 版，以及更新
提供的元件。它包含錯誤修正
和改進。

2022 年 3 月 23 日

[以 AWS IoT Device Tester 程
式設計方式下載](#)

新增如何 AWS IoT Greengrass
V2 以程式設計方式下載 IDT
for 的相關資訊。

2022 年 3 月 15 日

[Greengrass 開發套件 CLI 1.1.0 版](#)

Greengrass 開發套件 CLI 的 1.1.0 版已推出。此版本會將新的引數新增至 `component init` 和 `component publish` 命令。如果未建置元件，此版本也會更新 `component publish` 命令來建置元件。

2022 年 2 月 24 日

[Shadow Manager 2.1.0 版已發行](#)

Shadow Manager 元件 v2.1.0 已推出。此版本新增選項，以設定元件與影子同步的間隔 AWS IoT Core。例如，您可以指定較長的間隔，以減少頻寬用量和費用。

2022 年 2 月 3 日

[AWS IoT Greengrass Core 軟體 v2.5.3 的 Dockerfile 和 Docker 映像](#)

適用於 AWS IoT Greengrass Core 軟體 v2.5.3 的 Dockerfile 和 Docker 映像現已推出。

2022 年 1 月 12 日

[AWS IoT Device Tester v4.5.1 搭配發行的 GGV2Q v2.3.0](#)

IDT for AWS IoT Greengrass V2 4.5.1 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.3.0 版，並支援驗證和驗證使用硬體安全模組 (HSM) 來存放 AWS IoT Greengrass Core 軟體所用私有金鑰和憑證的 Linux 裝置。

2022 年 1 月 11 日

[AWS IoT Greengrass Core v2.5.3 軟體更新](#)

此版本提供 Greengrass 核元件的 2.5.3 AWS 版，以及更新提供的元件。它包含可讓您設定 AWS IoT Greengrass Core 軟體的支援，以使用您安全地存放在硬體安全模組 (HSM) 中的私有金鑰和憑證。

2022 年 1 月 6 日

[AWS IoT Greengrass Core
軟體 v2.5.2 的 Dockerfile 和
Docker 映像](#)

Core AWS IoT Greengrass
軟體 v2.5.2 的 Dockerfile 和
Docker 映像現已可用。

2021 年 12 月 20 日

[AWS IoT Device Tester v4.4.1
搭配發行的 GGV2Q v2.2.1](#)

IDT for AWS IoT Greengrass
V2 4.4.1 版已推出。此版本包
含 AWS IoT Greengrass V2 資
格套件 (GGV2Q) 2.2.1 版，並
支援 Greengrass 核 2.5.2 版的
裝置資格。

2021 年 12 月 12 日

[使用 Amazon Lookout for
Vision 執行機器學習推論](#)

新增如何在 Greengrass 核心
裝置上使用 Lookout for Vision
執行機器學習推論的相關資
訊。Lookout for Vision 使用電
腦視覺來尋找工業產品中的視
覺瑕疵。

2021 年 12 月 8 日

[AWS IoT Device Tester v4.4.1
搭配發行的 GGV2Q v2.2.0](#)

IDT for AWS IoT Greengrass
V2 4.4.1 版已推出。此版本包
含 AWS IoT Greengrass V2 資
格套件 (GGV2Q) 2.2.0 版，並
支援 Greengrass 核 2.5.2 版的
裝置資格。

2021 年 12 月 6 日

[AWS IoT Greengrass Core
v2.5.2 軟體更新](#)

此版本提供 Greengrass 核
元件的 2.5.2 AWS 版，以
及更新提供的元件。它修
正了 Greengrass 核更新後
Windows 服務的問題。它還
包含對 Windows 裝置上 AWS
IoT Device Defender 元件的支
援。

2021 年 12 月 3 日

[Kinesis Video Streams 元件的新邊緣連接器](#)

1.0.0 版的 Kinesis Video Streams 元件邊緣連接器已推出。此 AWS 提供的 會從本機攝影機讀取影片摘要，並將串流發佈至 Kinesis Video Streams。此元件與 AWS IoT TwinMaker 整合，可讓您檢視和管理 Grafana 儀表板中的影片串流和其他資料。

2021 年 11 月 30 日

[使用 管理 Greengrass 核心裝置 AWS Systems Manager](#)

新增如何使用 管理 Greengrass 核心裝置的相關資訊 AWS Systems Manager。Systems Manager 是一項 AWS 服務，可讓您檢視操作資料、自動化操作任務，以及維護安全性和合規性。

2021 年 11 月 29 日

[Greengrass 開發套件 CLI](#)

新增 AWS IoT Greengrass 開發套件命令列界面 (GDK CLI) 的相關資訊，這是您可以下載到本機開發電腦的工具，可協助您開發自訂 Greengrass 元件。您可以使用 GDK CLI 來建立、建置和發佈自訂元件。

2021 年 11 月 29 日

[社群提供的 Greengrass 元件](#)

新增 Greengrass 軟體目錄的相關資訊，這是一個由 Greengrass 社群開發的 Greengrass 元件索引。您可以從此目錄中下載、修改和部署元件，以建立 Greengrass 應用程式。

2021 年 11 月 29 日

[AWS IoT Greengrass Core v2.5.1 軟體更新](#)

此版本提供 Greengrass 核元件的 2.5.1 AWS 版，以及更新提供的元件。它包含對 Windows 裝置上的 32 位元 Java 的支援。它也修正了新物件群組移除行為和在 Windows 裝置上載入系統環境變數的問題。

2021 年 11 月 23 日

[AWS IoT Device Tester v4.4.0 搭配發行的 GGV2Q v2.1.0](#)

IDT for AWS IoT Greengrass V2 4.4.0 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.1.0 版，並支援執行 Greengrass 核 2.5.0 版的 Windows 型 Greengrass 裝置資格。

2021 年 11 月 19 日

[AWS IoT Greengrass Core v2.5.0 軟體更新](#)

此版本提供 Greengrass 核元件的 2.5.0 AWS 版，以及更新提供的元件。它包括在 Windows 裝置上執行 AWS IoT Greengrass 核心軟體的支援。它也會變更物件群組移除行為，並新增對 HTTPS 代理的支援。

2021 年 11 月 12 日

[SageMaker AI Edge Manager 1.1.0 版已發行](#)

Amazon SageMaker AI Edge Manager 元件 v1.1.0 已推出。此版本新增了對執行 Amazon Linux 2 的 Greengrass 核心裝置的支援，並新增了新的組態參數，以指定裝置上的擷取資料資料夾位置。

2021 年 11 月 3 日

[跨服務混淆代理人預防更新](#)

AWS IoT Greengrass V2 支援在 IAM 資源政策中使用 [aws:SourceArn](#) 和 [aws:SourceAccount](#) 全域條件內容金鑰，以防止混淆代理人問題。

2021 年 11 月 1 日

[用戶端裝置元件更新](#)

[用戶端裝置驗證](#) v2.0.3、[IP 偵測器](#) v2.1.0、[MQTT 橋接器](#) v2.1.0 和 [MQTT 代理程式 \(Moquette\)](#) v2.0.2 可供使用。此版本新增了對非預設 MQTT 代理程式連接埠的完整支援，並包含其他錯誤修正和改善。

2021 年 10 月 28 日

[Shadow Manager 2.0.4 版已發行](#)

Shadow Manager 元件 v2.0.4 已推出。此版本修正導致影子管理員刪除先前已刪除之任何影子新建立版本的問題。從此版本開始，DeleteThingShadow IPC 操作會遞增陰影版本。

2021 年 10 月 20 日

[Log Manager 2.2.0 版已發行](#)

Log Manager 元件 v2.2.0 已推出。Log Manager 現在支援使用組態映射來提供元件日誌組態。

2021 年 10 月 20 日

[Lambda Manager 2.1.4 版已發行](#)

Lambda Manager 元件 v2.1.4 已推出。此版本修正了導致使用 NodeJS 執行期的 Lambda 函數僅處理一則訊息的問題。

2021 年 10 月 20 日

[在 Docker 容器元件中使用程序間通訊、AWS 憑證和串流管理員](#)

新增如何在自訂 Docker 容器元件中使用程序間通訊 (IPC) AWS、登入資料和串流管理員的相關資訊。

2021 年 10 月 19 日

新的核遙測發射器元件	1.0.0 版的 核遙測發射器元件已推出。此 AWS 提供的元件會收集系統運作狀態遙測資料，並將其持續發佈至本機主題和 AWS IoT Core MQTT 主題。	2021 年 9 月 30 日
允許透過代理或防火牆的裝置流量	新增 Greengrass 核心裝置使用的端點和連接埠的相關資訊，因此您可以限制流量做為安全措施。	2021 年 9 月 16 日
AWS IoT Device Tester v4.2.0 搭配發行的 GGV2Q v2.0.1	IDT for AWS IoT Greengrass V2 4.2.0 版已更新為 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.0.1 版。此版本支援 Greengrass nucleus 2.4.0 版的裝置資格。	2021 年 8 月 31 日
已更新機器學習安裝程式元件	DLR 安裝程式元件 v1.6.5 和 TensorFlow Lite 安裝程式元件 v2.5.4 已推出。這些元件版本包含新的 UseInstaller 組態參數，可讓您停用預設安裝指令碼。	2021 年 8 月 30 日
的內嵌 Linux 支援 AWS IoT Greengrass	的 BitBake 配方 AWS IoT Greengrass V2 可在 GitHub 上的 meta-aws 專案中使用。您可以使用此配方，使用 Yocto 專案建置自訂的 Linux 作業系統。	2021 年 8 月 20 日
程式碼完整性	新增如何驗證 Greengrass AWS IoT Greengrass V2 核心裝置從下載之軟體完整性的相關資訊 AWS 雲端。	2021 年 8 月 19 日

VPC 端點 (AWS PrivateLink)	AWS IoT Greengrass 現在支援 AWS IoT Greengrass 控制平面的介面 VPC 端點 (AWS PrivateLink)。您可以在 VPC 與 AWS IoT Greengrass 控制平面之間建立私有連線。	2021 年 8 月 16 日
已發行串流管理員 v2.0.12	串流管理員 v2.0.12 現已推出。此版本修正了無法將串流管理員元件 2.0.7 版升級至 v2.0.8 和 v2.0.11 之間的版本的問題。	2021 年 8 月 10 日
AWS IoT Greengrass Core 軟體 v2.4.0 的 Dockerfile 和 Docker 映像	適用於 AWS IoT Greengrass Core 軟體 v2.4.0 的 Dockerfile 和 Docker 映像現已推出。	2021 年 8 月 9 日
AWS IoT Greengrass Core v2.4.0 軟體更新	此版本提供 Greengrass 核元件的 2.4.0 AWS 版，以及更新提供的元件。它包括對元件系統資源限制的支援、暫停和恢復元件的 IPC 操作，以及佈建外掛程式。	2021 年 8 月 3 日
新 AWS IoT SiteWise 元件	新增下列 AWS 提供的元件 AWS IoT SiteWise： IoT SiteWise OPC-UA 收集器 、 IoT SiteWise 發佈者 和 IoT SiteWise 處理器 。	2021 年 7 月 29 日
AWS IoT Device Tester 發行 GGV2Q 2.0.0 版的 v4.2.0	IDT AWS IoT Greengrass forV2 4.2.0 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 2.0.0 版，並支援 Docker 元件、機器學習和串流管理員的選用資格測試。	2021 年 7 月 14 日

AWS IoT Greengrass AWS IoT Device SDK 適用於 C++ v2 的核心 IPC 程式庫	AWS IoT Device SDK 適用於 C++ v2 的 1.13.0 版支援 AWS IoT Greengrass Core IPC，因此您可以在 C++ 中開發與 AWS IoT Greengrass Core 軟體互動的元件。	2021 年 7 月 14 日
SageMaker AI Edge Manager 元件 1.0.2 版已發行	Amazon SageMaker AI Edge Manager 元件 v1.0.2 已推出。此版本會更新元件生命週期中的安裝指令碼。您的核心裝置現在必須在裝置上安裝 Python 3.6 或更新版本，包括 pip 適用於您 Python 版本的，才能部署此元件。	2021 年 7 月 12 日
AWS IoT Device Tester for AWS IoT Greengrass V2 的支援更新	IDT for AWS IoT Greengrass V2 4.1.0 版現在支援使用 Greengrass 核 2.3.0 版來取得裝置資格。	2021 年 7 月 8 日
AWS IoT Greengrass Core 軟體 v2.3.0 的 Dockerfile 和 Docker 映像	Core AWS IoT Greengrass 軟體 v2.3.0 的 Dockerfile 和 Docker 映像現已推出。	2021 年 7 月 7 日
AWS 受管政策	新增 AWS 受管政策的相關資訊 AWS IoT Greengrass。	2021 年 7 月 2 日
新的建議 JVM 選項	新增建議 JVM 選項的相關資訊，以控制 AWS IoT Greengrass Core 軟體的記憶體配置。	2021 年 6 月 30 日
AWS IoT Greengrass 核心 v2.3.0 軟體更新	此版本提供 Greengrass 核元件的 2.3.0 AWS 版，以及更新提供的元件。它包含對部署中大型元件組態文件的支援。	2021 年 6 月 29 日

AWS IoT Greengrass Core 軟體 v2.2.0 的 Dockerfile 和 Docker 映像	Core AWS IoT Greengrass 軟體 v2.2.0 的 Dockerfile 和 Docker 映像現已可用。	2021 年 6 月 28 日
AWS IoT Device Tester v4.1.0 搭配發行的 GGV2Q v1.1.1	IDT for AWS IoT Greengrass V2 4.1.0 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 1.1.1 版，並支援使用 Greengrass 核 2.2.0、2.1.0 版和 2.0.5 版來取得裝置資格。	2021 年 6 月 18 日
AWS IoT Greengrass Core v2.2.0 軟體更新	此版本提供 Greengrass 核元件的 2.2.0 AWS 版，以及更新提供的元件。它包含您可以部署的元件，以新增對用戶端裝置的支援，並新增本機影子服務。	2021 年 6 月 18 日
Lambda 啟動器 2.0.6 版已發行	Lambda 啟動器元件的 2.0.6 版已推出。此版本包含效能改善和錯誤修正。	2021 年 6 月 13 日
已發行新的 SageMaker AI Edge Manager 元件	適用於 Amazon SageMaker AI Edge Manager 元件的 1.0.0 版 AWS IoT Greengrass。此元件會在 Greengrass 核心裝置上安裝 SageMaker AI Edge Manager 代理程式二進位檔。	2021 年 6 月 10 日
元件類型	新增 中元件類型的相關資訊 AWS IoT Greengrass。元件類型指定 AWS IoT Greengrass Core 軟體如何執行元件。	2021 年 6 月 3 日

[AWS IoT Device Tester v4.0.2 搭配發行的 GGV2Q v1.1.0](#)

IDT AWS IoT Greengrass V2 版已推出。此版本包含 AWS IoT Greengrass V2 資格套件 (GGV2Q) 1.1.0 版，並支援搭配 Greengrass CLI 2.1.0 版使用 Greengrass 核。這也包括 MQTT 和 Lambda 的新必要測試群組，以及其他次要錯誤修正和改善。

2021 年 5 月 5 日

[AWS IoT Greengrass Core 軟體 v2.1.0 的 Dockerfile 和 Docker 映像](#)

Core AWS IoT Greengrass 軟體 v2.1.0 的 Dockerfile 和 Docker 映像現已可用。Docker 映像可讓您在 Amazon Linux 2 做為基礎作業系統的 Docker 容器中執行 AWS IoT Greengrass Core 軟體。

2021 年 4 月 27 日

[AWS IoT Greengrass Core v2.1.0 軟體更新](#)

此版本提供 Greengrass 核元件的 2.1.0 AWS 版，以及更新提供的元件。其中包括可用來從私有 Amazon ECR 儲存庫下載 Docker 映像的新元件，以及使用 TensorFlow Lite 執行機器學習推論的新範例元件。

2021 年 4 月 26 日

[使用 Secrets Manager 的範例 元件](#)

新增了一個範例元件，可列印您部署到核心裝置的 AWS Secrets Manager 秘密值。

2021 年 4 月 8 日

[Greengrass 核心裝置的最低 AWS IoT 政策](#)

新增在核心裝置上支援基本 Greengrass 功能所需的最低許可集資訊。

2021 年 4 月 2 日

訂閱 IPC 事件串流	新增如何使用程序間通訊 (IPC) 操作來訂閱 Greengrass 核心裝置上的事件串流的相關資訊。	2021 年 4 月 1 日
AWS IoT Device Tester 的支援更新 AWS IoT Greengrass	IDT for AWS IoT Greengrass V2 4.0.1 版現在支援使用 Greengrass 核 2.0.5 版搭配 Greengrass CLI 2.0.5 版來取得裝置資格。	2021 年 3 月 17 日
建立使用串流管理員的自訂元件	新增如何設定元件配方和成品的資訊，以開發管理資料串流的應用程式。	2021 年 3 月 9 日
AWS IoT Greengrass Core v2.0.5 軟體更新	此版本提供 Greengrass 核元件的 2.0.5 AWS 版，以及更新提供的元件。它修正了網路代理支援的問題，以及 AWS 中國區域中 Greengrass 資料平面端點的問題。	2021 年 3 月 9 日
元件環境變數參考	新增 AWS IoT Greengrass Core 軟體為元件設定的環境變數相關資訊。您可以使用這些環境變數來取得物件名稱 AWS 區域，以及 Greengrass 核版本。	2021 年 2 月 23 日

[手動安裝](#)

新增如何手動建立必要 AWS 資源，或在防火牆或網路代理後方安裝的相關資訊。透過使用手動安裝，您不需要授予安裝程式在中建立資源的許可 AWS 帳戶，因為您建立必要的 AWS IoT 和 IAM 資源。您也可以將裝置設定為在連接埠 443 上或透過網路代理進行連線。

2021 年 2 月 17 日

[AWS IoT Greengrass AWS IoT Device SDK 適用於 Python v2 的核心 IPC 程式庫更新](#)

AWS IoT Device SDK 適用於 Python v2 的 1.5.4 版簡化了連線至 AWS IoT Greengrass Core IPC 服務所需的步驟。

2021 年 2 月 11 日

[AWS IoT Device Tester 的支援更新 AWS IoT Greengrass](#)

IDT for AWS IoT Greengrass V2 4.0.1 版現在支援使用 Greengrass 核 2.0.4 版搭配 Greengrass CLI 2.0.4 版來取得裝置資格。

2021 年 2 月 5 日

[匯入 Lambda 函數的新教學課程](#)

新增以主控台為基礎的教學課程，將 Lambda 函數匯入為在 Greengrass 核心裝置上執行的元件。

2021 年 2 月 5 日

[AWS IoT Greengrass Core v2.0.4 軟體更新](#)

此版本提供 Greengrass 核元件的 2.0.4 版。它包含新 greengrassDataPlanePort 參數，以透過連接埠 443 設定 HTTPS 通訊並修正錯誤。使用執行 AWS IoT Greengrass Core 軟體安裝程式 `sts:GetCallerIdentity` 時，最小 IAM 政策現在需要 `iam:GetPolicy` 和 `--provision true`。

2021 年 2 月 4 日

[發佈新的安全通道元件](#)

提供 1.0.0 版的安全通道元件 AWS IoT Greengrass。此 AWS 提供的元件使用 AWS IoT 安全通道，與受限防火牆後方的 Greengrass 核心裝置建立安全的雙向通訊。

2021 年 1 月 21 日

[AWS IoT Device Tester for AWS IoT Greengrass v4.0.1 已發行](#)

IDT for AWS IoT Greengrass V2 4.0.1 版已推出。此版本可讓您使用 IDT 來開發和執行自訂測試套件，以進行裝置驗證。這也包含 macOS 和 Windows 的程式碼簽署 IDT 應用程式。

2020 年 12 月 22 日

[的初始版本 AWS IoT Greengrass Version 2](#)

AWS IoT Greengrass V2 是的新主要版本版本 AWS IoT Greengrass。此版本新增了數種功能，例如模組化軟體元件和持續部署。這些功能可讓您更輕鬆地開發和管理邊緣應用程式。

2020 年 12 月 15 日

AWS 詞彙表

如需最新的 AWS 術語，請參閱 AWS 詞彙表 參考中的[AWS 詞彙表](#)。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。