



Entwicklerhandbuch, Version 1

AWS IoT Greengrass



AWS IoT Greengrass: Entwicklerhandbuch, Version 1

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

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

.....	xxi
Was ist AWS IoT Greengrass?	1
AWS IoT Greengrass Kernsoftware	3
AWS IoT Greengrass Kern-Softwareversionen	4
AWS IoT Greengrass Gruppen	15
Geräte in AWS IoT Greengrass	17
SDKs	20
Unterstützte Plattformen und Anforderungen	21
AWS IoT Greengrass lädt herunter	35
AWS IoT Greengrass Kernsoftware	35
AWS IoT Greengrass Snap-Software	44
AWS IoT Greengrass Docker-Software	44
AWS IoT Greengrass Kern-SDK	47
Unterstützte Machine Learning-Laufzeiten und -Bibliotheken	47
AWS IoT Greengrass ML SDK-Software	48
Bitte geben Sie uns Feedback	49
Installieren Sie die AWS IoT Greengrass Core-Software	49
Herunterladen und Extrahieren einer tar.gz-Datei	50
Ausführen des Greengrass Device Setup-Skripts	50
Installieren aus einem APT-Repository	50
AWS IoT Greengrass In einem Docker-Container ausführen	53
Lauf AWS IoT Greengrass im Handumdrehen	53
Archivieren einer Core-Software-Installation	65
Konfiguriere den AWS IoT Greengrass Kern	67
AWS IoT Greengrass Kernkonfigurationsdatei	68
Dienstendpunkte müssen dem Zertifikatstyp entsprechen	134
Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy	135
Konfigurieren eines Schreibverzeichnisses	146
Konfigurieren der MQTT-Einstellungen	150
Aktivieren der automatischen IP-Erkennung	169
Ausführen von Greengrass bei Systemstart	173
Weitere Informationen finden Sie auch unter	175
AWS IoT Greengrass V1 Wartungspolitik	176
AWS IoT Greengrass Versionsschema	176

Lebenszyklusphasen für die AWS IoT Greengrass Core-Software	177
Wartungsrichtlinie für die AWS IoT Greengrass Core-Software	177
Zeitplan für die Wartungsphase	178
Einstellungszeitplan	178
Support-Richtlinie für Lambda-Funktionen	179
Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1	179
Ende des Wartungsplans	179
Ende der Wartung für AWS IoT Greengrass Docker-Images der Core-Software v1.x	45
Ende der Wartung für das APT-Repository der AWS IoT Greengrass Core-Software v1.x ...	181
Ende der Wartung für die AWS IoT Greengrass Core-Software v1.11.x Snap	181
Erste Schritte mit AWS IoT Greengrass	182
Auswahl der ersten Schritte	182
Voraussetzungen	185
Erstelle eine AWS-Konto	187
Melde dich an für ein AWS-Konto	187
Erstellen eines Benutzers mit Administratorzugriff	187
Schnellstart: Greengrass-Geräteeinrichtung	189
Voraussetzungen	190
Ausführen von Greengrass Device Setup	190
Beheben von -Problemen	194
Konfigurationsoptionen für Greengrass Device Setup	195
Modul 1: Umgebungseinrichtung für Greengrass	206
Einrichten eines Raspberry Pi	207
Eine EC2 Amazon-Instance einrichten	216
Einrichten anderer Geräte	221
Modul 2: Installation der AWS IoT Greengrass Core-Software	225
Bereitstellung AWS IoT eines Objekts, das als Greengrass-Kern verwendet werden kann ...	226
Eine Greengrass-Gruppe erstellen	229
Installation und Ausführung AWS IoT Greengrass auf dem Kerngerät	230
Modul 3 (Teil 1): Lambda-Funktionen auf AWS IoT Greengrass	238
Eine Lambda-Funktion erstellen und verpacken	239
Konfigurieren Sie die Lambda-Funktion für AWS IoT Greengrass	244
Bereitstellen von Cloud-Konfigurationen für ein Core-Gerät	248
Stellen Sie sicher, dass die Lambda-Funktion auf dem Kerngerät ausgeführt wird	249
Modul 3 (Teil 2): Lambda-Funktionen auf AWS IoT Greengrass	250
Lambda-Funktion erstellen und verpacken	251

Konfigurieren Sie langlebige Lambda-Funktionen für AWS IoT Greengrass	255
Testen Sie langlebige Lambda-Funktionen	256
Testen Sie Lambda-Funktionen auf Abruf	259
Modul 4: Interaktion mit Client-Geräten in einer AWS IoT Greengrass Gruppe	263
Client-Geräte in einer AWS IoT Greengrass Gruppe erstellen	265
Konfigurieren von Abonnements	268
Installieren Sie das AWS IoT Device SDK für Python	269
Testen der Kommunikation	276
Modul 5: Interaktion mit Geräteschatten	280
Konfigurieren von Geräten und Abonnements	282
Download der benötigten Dateien	284
Testen der Kommunikation (Gerätesynchronisierungen deaktiviert)	285
Testen der Kommunikation (Gerätesynchronisierungen aktiviert)	289
Modul 6: Zugriff auf andere AWS Dienste	290
Konfigurieren der Gruppenrolle	292
Die Lambda-Funktion erstellen und konfigurieren	294
Konfigurieren von Abonnements	297
Testen der Kommunikation	298
Modul 7: Simulation der Hardware-Sicherheitsintegration	300
Installieren von SoftHSM	301
Konfigurieren von SoftHSM	302
Importieren des privaten Schlüssels	303
Konfigurieren des Greengrass Cores	304
Testen der Konfiguration	308
Weitere Informationen finden Sie auch unter	309
OTA-Updates der AWS IoT Greengrass Core-Software	310
Voraussetzungen	310
IAM-Berechtigungen für OTA-Updates	312
Überlegungen	314
Greengrass OTA-Update-Agent	315
Integration in Init-Systeme	316
Managed Respawn mit OTA-Updates	317
Erstellen eines OTA-Updates	319
CreateSoftwareUpdateJob API	322
Stellen Sie AWS IoT Greengrass Gruppen bereit	325
Bereitstellen von Gruppen (Konsole)	326

Bereitstellen von Gruppen (API)	328
Abrufen der Gruppen-ID	329
Übersicht über das Gruppenobjektmodell	331
Gruppen	331
Gruppenversionen	331
Gruppenkomponenten	332
Aktualisieren von Gruppen	333
Weitere Informationen finden Sie auch unter	335
Abrufen von Bereitstellungsbenachrichtigungen	335
Änderungsereignis für den Gruppenbereitstellungsstatus	336
Voraussetzungen für die Erstellung von EventBridge Regeln	338
Konfigurieren von Bereitstellungsbenachrichtigungen (Konsole)	338
Konfigurieren von Bereitstellungsbenachrichtigungen (CLI)	340
Konfigurieren von Bereitstellungsbenachrichtigungen (CloudFormation)	341
Weitere Informationen finden Sie auch unter	341
Zurücksetzen von Bereitstellungen	341
Setzen Sie Bereitstellungen von der Konsole aus zurück AWS IoT	342
Setzen Sie Bereitstellungen mit der API zurück AWS IoT Greengrass	342
Weitere Informationen finden Sie auch unter	344
Erstellen von Sammelbereitstellungen	344
Voraussetzungen	345
Erstellen und Hochladen der Eingabedatei für die Sammelbereitstellung	345
Erstellen und konfigurieren Sie eine IAM-Ausführungsrolle für Massenbereitstellungen	348
Ermöglichen des Zugriff auf Ihren S3-Bucket für Ihre Ausführungsrolle	350
Bereitstellen der Gruppen	352
Testen der Bereitstellung	355
Fehlerbehebung bei Sammelbereitstellungen	357
Weitere Informationen finden Sie auch unter	359
Lokale Lambda-Funktionen ausführen	360
SDKs	361
Migration von Cloud-basierten Lambda-Funktionen	365
Referenzieren von Funktionen nach Alias oder Version	366
Steuerung der Ausführung von Greengrass Lambda-Funktionen	366
Gruppenspezifische Konfigurationseinstellungen	366
Eine Lambda-Funktion als Root ausführen	371
Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen	373

Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe	377
Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen	379
Kommunikationsströme	380
Kommunikation über MQTT-Nachrichten	380
Andere Kommunikationsströme	381
Abrufen des Eingabethemas (oder -betreffs)	382
Lebenszyklus-Konfiguration	384
Ausführbare Lambda-Dateien	386
Eine ausführbare Lambda-Datei erstellen	387
AWS IoT Greengrass In einem Docker-Container ausführen	389
Voraussetzungen	390
Holen Sie sich das AWS IoT Greengrass Container-Image von Amazon ECR	392
Erstellen und Konfigurieren von Greengrass-Gruppe und Core	395
AWS IoT Greengrass Lokal ausführen	396
Konfigurieren der "Kein Container"-Containerisierung für die Gruppe	399
Stellen Sie Lambda-Funktionen im Docker-Container bereit	400
(Optional) Stellen Sie Client-Geräte, die mit Greengrass interagieren, im Docker-Container bereit	401
Den AWS IoT Greengrass Docker-Container stoppen	401
Problembehandlung in einem Docker-Container AWS IoT Greengrass	401
Zugreifen auf lokale Ressourcen	405
Unterstützte Ressourcentypen	405
Voraussetzungen	407
Volume-Ressourcen im /proc-Verzeichnis	407
Dateizugriffsberechtigung des Gruppenbesitzers	408
Weitere Informationen finden Sie auch unter	408
Verwenden der -CLI	409
Erstellen lokaler Ressourcen	409
Erstellen der Greengrass-Funktion	411
Fügen Sie die Lambda-Funktion zur Gruppe hinzu	413
Fehlerbehebung	415
Verwenden der Konsole	416
Voraussetzungen	417
Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen	417
Eine Lambda-Funktion erstellen und veröffentlichen	419
Fügen Sie die Lambda-Funktion zur Gruppe hinzu	421

Hinzufügen einer lokalen Ressource zur Gruppe	422
Hinzufügen von Abonnements zur Gruppe	423
Bereitstellen der Gruppe	424
Testen des lokalen Ressourcenzugriffs	426
Durchführen von Machine Learning-Inferenzen	429
Wie funktioniert AWS IoT Greengrass ML-Inferenz	429
Machine Learning-Ressourcen	430
Unterstützte Modellquellen	430
Voraussetzungen	433
Laufzeiten und Bibliotheken für ML-Inferenz	433
SageMaker Deep-Learning-Laufzeit von AI Neo	434
MXNet Versionierung	434
MXNet auf Raspberry Pi	435
TensorFlow Einschränkungen bei der Modellverwaltung auf dem Raspberry Pi	435
Zugreifen auf Machine Learning-Ressourcen	436
Zugriffsberechtigungen für Machine Learning-Ressourcen	436
Zugriffsberechtigungen für Lambda-Funktionen definieren (Konsole)	439
Definition von Zugriffsberechtigungen für Lambda-Funktionen (API)	440
Zugreifen auf Ressourcen für maschinelles Lernen über den Lambda-Funktionscode	443
Fehlerbehebung	444
Weitere Informationen finden Sie auch unter	447
So konfigurieren Sie Machine Learning-Inferenz	447
Voraussetzungen	448
Konfigurieren des Raspberry Pi	449
Installieren Sie das MXNet Framework	451
Erstellen eines Modellpakets	451
Eine Lambda-Funktion erstellen und veröffentlichen	452
Fügen Sie die Lambda-Funktion zur Gruppe hinzu	455
Hinzufügen von Ressourcen zur Gruppe	457
Hinzufügen eines Abonnements zur Gruppe	460
Bereitstellen der Gruppe	461
Testen der Anwendung	462
Nächste Schritte	466
Konfigurieren eines Intel Atom	466
Konfiguration eines NVIDIA Jetson TX2	470
So konfigurieren Sie optimierte Machine Learning-Inferenz	475

Voraussetzungen	448
Konfigurieren des Raspberry Pi	477
Installieren Sie die Neo Deep Learning Runtime	479
Erstellen Sie eine Inferenz-Lambda-Funktion	479
Fügen Sie die Lambda-Funktion zur Gruppe hinzu	483
Hinzufügen einer Neo-optimierten Modellressource zur Gruppe	485
Hinzufügen Ihrer Kamerageräte-Ressource zur Gruppe	487
Hinzufügen von Abonnements zur Gruppe	489
Bereitstellen der Gruppe	489
Testen des Beispiels	490
Konfigurieren eines Intel Atom	491
Konfiguration eines NVIDIA Jetson TX2	495
Problembehandlung bei AWS IoT Greengrass ML-Inferenz	463
Nächste Schritte	502
Verwalten von Daten-Streams	503
Stream-Management-Workflow	504
Voraussetzungen	506
Datensicherheit	507
Lokale Datensicherheit	507
Client-Authentifizierung	508
Weitere Informationen finden Sie auch unter	509
Konfigurieren des Stream-Managers	509
Stream-Manager-Parameter	510
Konfigurieren der Einstellungen (Konsole)	513
Konfigurieren der Einstellungen (CLI)	516
Weitere Informationen finden Sie auch unter	526
Wird verwendet StreamManagerClient , um mit Streams zu arbeiten	527
Erstellen eines Nachrichten-Streams	528
Anhängen einer Nachricht	532
Lesen von Nachrichten	539
Auflisten von Streams	541
Beschreiben eines Nachrichten-Streams	543
Nachrichtenstream aktualisieren	545
Löschen eines Nachrichten-Streams	550
Weitere Informationen finden Sie auch unter	551
Konfigurationen für unterstützte AWS Cloud Ziele exportieren	551

Exportieren von Daten-Streams (Konsole)	569
Voraussetzungen	570
Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen	572
Erstellen einer Lambda-Funktion	576
Hinzufügen einer Funktion zur Gruppe	578
Aktivieren des Stream-Managers	579
Konfigurieren der lokalen Protokollierung	579
Bereitstellen der Gruppe	580
Testen der Anwendung	581
Weitere Informationen finden Sie auch unter	583
Exportieren von Daten-Streams (CLI)	583
Voraussetzungen	584
Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen	587
Erstellen einer Lambda-Funktion	590
Erstellen einer Funktionsdefinition und -version	592
Erstellen einer Logger-Definition und -Version	594
Abrufen des ARN der Core-Definitionsversion	595
Erstellen einer Gruppenversion	597
Eine Bereitstellung auswählen	597
Testen der Anwendung	599
Weitere Informationen finden Sie auch unter	600
Bereitstellen von Secrets für den Core	601
Verschlüsselung von Secrets	603
Voraussetzungen	604
Angabe des privaten Schlüssels für die Verschlüsselung von Secrets	605
Erlaubt das Abrufen geheimer Werte AWS IoT Greengrass	606
Weitere Informationen finden Sie auch unter	608
Arbeiten mit geheimen Ressourcen	608
Erstellen und Verwalten von Secrets	609
Verwenden von lokalen Secrets	614
Erstellen einer geheimen Ressource (Konsole)	617
Voraussetzungen	618
Ein Secrets Manager Manager-Geheimnis erstellen	619
Hinzufügen einer geheimen Ressource zu einer Gruppe	620
Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen	621
Erstellen einer Lambda-Funktion	623

Hinzufügen der -Funktion zur Gruppe	625
Anfügen der geheimen Ressource an die Funktion	627
Hinzufügen von Abonnements zur Gruppe	628
Bereitstellen der Gruppe	628
Lambda-Funktion testen	630
Weitere Informationen finden Sie auch unter	631
Integrieren von Services und Protokollen mit Konnektoren	632
Voraussetzungen	633
Verwenden von Greengrass-Konnektoren	634
Konfigurationsparameter	636
Parameter für den Zugriff auf Gruppenressourcen	637
Aktualisieren von Konnektorparametern	637
Eingaben und Ausgaben	638
Eingabethemen	639
Unterstützung der Containerisierung	640
Aktualisieren von Konnektorversionen	640
Protokollierung	641
AWS-mitgelieferte Greengrass-Anschlüsse	642
CloudWatch Metriken	645
Device Defender	662
Docker-Anwendungsbereitstellung	668
IoT Analytics	714
IoT-Ethernet-IP-Protokolladapter	731
IoT SiteWise	736
Kinesis Firehose	753
ML-Feedback	772
ML-Bildklassifizierung	790
ML-Objekterkennung	817
Modbus-RTU-Protokolladapter	835
Modbus-TCP-Protokolladapter	854
Raspberry Pi GPIO	860
Serieller Stream	871
ServiceNow MetricBase Integration	885
SNS	901
Splunk-Integration	914
Twilio-Benachrichtigungen	928

Beginnen Sie mit Konnektoren (Konsole)	946
Voraussetzungen	948
Erstellen Sie ein Secrets Manager Manager-Geheimnis	949
Hinzufügen einer geheimen Ressource zu einer Gruppe	950
Hinzufügen eines Konnektors zur Gruppe	951
Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen	952
Erstellen einer Lambda-Funktion	953
Hinzufügen einer Funktion zur Gruppe	955
Hinzufügen von Abonnements zur Gruppe	956
Bereitstellen der Gruppe	957
Testen der Lösung	958
Weitere Informationen finden Sie auch unter	959
Erste Schritte mit Konnektoren (CLI)	960
Voraussetzungen	962
Ein Secrets Manager Manager-Geheimnis erstellen	963
Erstellen einer Ressourcendefinition und -version	964
Erstellen einer Konnektordefinition und -version	965
Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen	966
Erstellen einer Lambda-Funktion	968
Erstellen einer Funktionsdefinition und -version	970
Erstellen einer Abonnementdefinition und -version	971
Erstellen einer Gruppenversion	972
Eine Bereitstellung auswählen	974
Testen der Lösung	976
Weitere Informationen finden Sie auch unter	977
Greengrass Discovery-API RESTful	978
Anforderung	978
Antwort	979
Discovery-Autorisierung	980
Beispieldokumente für Discovery-Antwort	980
Sicherheit	983
Überblick über die AWS IoT Greengrass Sicherheit	984
Geräteverbindung – Workflow	985
AWS IoT Greengrass Sicherheit konfigurieren	986
Sicherheitsprinzipale	987
Verwaltete Abonnements im MQTT Messaging-Workflow	991

Support für TLS-Verschlüsselungs-Suites	991
Datenschutz	994
Datenverschlüsselung	995
Integration von Hardware-Sicherheit	998
Geräteauthentifizierung und -autorisierung	1020
X.509-Zertifikate	1021
AWS IoT Richtlinien	1024
Minimale AWS IoT Richtlinie für das Kerngerät	1026
Identity and Access Management	1031
Zielgruppe	1031
Authentifizierung mit Identitäten	1031
Verwalten des Zugriffs mit Richtlinien	1033
Weitere Informationen finden Sie auch unter	1034
Wie AWS IoT Greengrass funktioniert mit IAM	1035
Greengrass-Servicerolle	1043
Greengrass-Gruppenrolle	1052
Serviceübergreifende Confused-Deputy-Prävention	1062
Beispiele für identitätsbasierte Richtlinien	1063
Problembehandlung bei Identitäts- und Zugriffsproblemen	1067
Compliance-Validierung	1070
Ausfallsicherheit	1070
Sicherheit der Infrastruktur	1071
Konfigurations- und Schwachstellenanalyse	1072
VPC-Endpunkte (AWS PrivateLink)	1073
Überlegungen zu AWS IoT Greengrass VPC-Endpunkten	1074
Erstellen Sie einen VPC-Schnittstellen-Endpunkt für den Betrieb auf AWS IoT Greengrass	
Steuerungsebene	1074
Erstellen einer VPC-Endpunktrichtlinie für AWS IoT Greengrass	1075
Bewährte Methoden für die Gewährleistung der Sicherheit	1076
Erteilen von Mindestberechtigungen	1076
Hardcodieren Sie Anmeldeinformationen in Lambda-Funktionen nicht fest	1076
Keine Protokollierung sensibler Informationen	1076
Erstellen gezielter Abonnements	1077
Synchronisieren der internen Uhr Ihres Geräts	1077
Verwalten der Geräteauthentifizierung mit dem Greengrass Core	1077
Weitere Informationen finden Sie auch unter	1079

Protokollierung und Überwachung	1080
Überwachungstools	1080
Weitere Informationen finden Sie auch unter	1081
Überwachen mit AWS IoT Greengrass -Protokollen	1081
Zugreifen auf Protokolle CloudWatch	1082
Zugreifen auf Dateisystemprotokolle	1084
Standardkonfiguration für die Protokollierung	1085
Konfigurieren Sie die Protokollierung für AWS IoT Greengrass	1085
Einschränkungen für die Protokollierung	1089
CloudTrail Logs	1090
AWS IoT Greengrass API-Aufrufe protokollieren mit AWS CloudTrail	1090
AWS IoT Greengrass Informationen in CloudTrail	1091
AWS IoT Greengrass Logdateieinträge verstehen	1092
Weitere Informationen finden Sie auch unter	1095
Erfassung von Telemetriedaten zur Systemintegrität	1096
Konfiguration der Telemetrieinstellungen	1099
Abonnieren Sie den Empfang von Telemetriedaten	1103
Fehlerbehebung bei AWS IoT Greengrass Telemetrie	1110
Aufruf der lokalen Health Check-API	1110
Holen Sie sich Gesundheitsinformationen für alle Arbeitnehmer	1111
Ruft Gesundheitsinformationen zu bestimmten Mitarbeitern ab	1112
Informationen zur Gesundheit der Arbeitnehmer	1114
Markieren Ihrer Greengrass-Ressourcen	1118
Grundlagen zu Tags (Markierungen)	1118
Markierungsunterstützung (Konsole)	1118
Markierungsunterstützung (API)	1119
Verwenden von Tags mit IAM-Richtlinien	1121
IAM-Beispielrichtlinien	1122
Weitere Informationen finden Sie auch unter	1124
CloudFormation Unterstützung für AWS IoT Greengrass	1125
Erstellen von -Ressourcen	1125
Bereitstellen von Ressourcen	1126
-Beispielvorlage	1127
Unterstützt s AWS-Region	1140
Verwenden von AWS IoT Device Tester für AWS IoT Greengrass V1	1141
AWS IoT Greengrass Qualifizierungssuite	1141

Benutzerdefinierte Testsuiten	1142
Unterstützte Versionen von AWS IoT Device Tester für AWS IoT Greengrass V1	1142
Nicht unterstützte IDT-Versionen für AWS IoT Greengrass	1143
Verwenden Sie IDT, um die AWS IoT Greengrass Qualification Suite auszuführen	1149
Test-Suite-Versionen	1150
Beschreibung der Testgruppen	1151
Voraussetzungen	1155
Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests	1166
IDT-Einstellungen konfigurieren	1190
Führen Sie die AWS IoT Greengrass Qualifizierungssuite aus	1205
Verstehen von Ergebnissen und Protokollen	1210
Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen	1214
Herunterladen der neuesten Version von IDT für AWS IoT Greengrass	1156
Workflow zur Erstellung einer Testsuite	1215
Tutorial: IDT-Beispiel-Testsuite erstellen und ausführen	1216
Tutorial: Entwickeln Sie eine einfache IDT-Testsuite	1221
Erstellen Sie Konfigurationsdateien für die IDT-Testsuite	1231
Konfigurieren Sie die IDT-Zustandsmaschine	1239
Ausführbare IDT-Testfalldateien erstellen	1263
Verwenden Sie den IDT-Kontext	1271
Einstellungen für Testläufer konfigurieren	1276
Debuggen und Ausführen benutzerdefinierter Testsuiten	1288
Überprüfen Sie die IDT-Testergebnisse und -protokolle	1291
IDT-Nutzungsmetriken	1297
IDT zur Fehlerbehebung AWS IoT Greengrass	1304
Fehlercodes	1305
IDT auf Fehler beheben AWS IoT Greengrass	1329
Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1	1334
Fehlerbehebung	1335
AWS IoT Greengrass Kernprobleme	1335
Fehler: In der Konfigurationsdatei fehlt das CaPath, CertPath oder KeyPath. Der Greengrass-Daemon-Prozess mit [pid = <pid>] ist abgestürzt.	1338
Fehler: Fehler beim Analysieren von /<greengrass-root>/config/config.json.	1338
Fehler: Beim Generieren der TLS-Konfiguration ist ein Fehler aufgetreten: ErrUnknown URIScheme	1339

Fehler: Laufzeit konnte nicht gestartet werden: Es konnten keine Worker gestartet werden: Zeitüberschreitung für den Container-Test.	1339
<address>Fehler: Fehler beim Aufrufen PutLogEvents auf lokaler Cloudwatch, LogGroup:/ GreengrassSystem/connection_manager, Fehler:: Sendeanforderung ist fehlgeschlagen, verursacht durch RequestError: Post http://<path>/cloudwatch/logs/: dial tcp: getsockopt: Verbindung verweigert, Antwort: {}.	1340
<account-id><function-name><version><file-name>Fehler: Server konnte nicht erstellt werden, weil: Gruppe konnte nicht geladen werden: chmod/<greengrass- root>/:aws:lambda: ::function::/ggc/deployment/lambda/arn: keine solche Datei oder kein <region>solches Verzeichnis.	1340
Die AWS IoT Greengrass Core-Software startet nicht, nachdem Sie von der Ausführung ohne Containerisierung zur Ausführung in einem Greengrass-Container gewechselt sind. .	1341
Fehler: Spulengröße sollte mindestens 262144 Bytes betragen.	1341
Fehler: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {„errorString“: „operation timed out“} (Cloud Messaging Fehler: Fehler trat beim Versuch zur Veröffentlichung einer Nachricht auf)	1341
<version>Fehler: container_linux.go:344: Das Starten des Container-Prozesses verursachte „process_linux.go:424: container init caused\" rootfs_linux.go:64: mount\\\"\"_ipc.sock: permission denied\\\"\". greengrass/ggc/socket/greengrass_ipc.sock\\\"\" to rootfs \\\"\"/ greengrass/ggc/packages rootfs/merged\\\"\" at \\\"\"/greengrass_ipc.sock\\\"\" caused \\\"\"stat / greengrass/ggc/socket/greengrass	1342
Fehler: Greengrass-Daemon wird mit folgender PID ausgeführt: <process-id>. Einige Systemkomponenten konnten nicht gestartet werden. Überprüfen Sie die Datei „runtime.log“ auf Fehler.	1342
Der Geräteschatten wird nicht mit der Cloud synchronisiert.	1068
FEHLER: TCP-Verbindung kann nicht angenommen werden. accept tcp [::]:8000: accept4: zu viele geöffnete Dateien.	1343
Fehler: Laufzeit-Ausführungsfehler: Lambda-Container kann nicht gestartet werden. container_linux.go:259: starting container process caused \"process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"\"permission denied\\\"\"\".	1344
Warnung: [WARN] - [5] GK Remote: Fehler beim Abrufen der öffentlichen Schlüsseldaten ErrPrincipalNotConfigured: Der private Schlüssel für MqttCertificate ist nicht festgelegt.	1344
<account-id><role-name><region>Fehler: Beim Versuch, die Rolle arn:aws:iam: :role/ für den Zugriff auf die S3-URL https://-greengrass-updates.s3 zu verwenden, wurde die Berechtigung verweigert. <region><architecture><distribution-version>.amazonaws.com/ core/ /greengrass-core- .tar.gz.	1068

Der AWS IoT Greengrass Core ist für die Verwendung eines Netzwerk-Proxys konfiguriert und Ihre Lambda-Funktion kann keine ausgehenden Verbindungen herstellen.	1345
Der Core befindet sich in einer unendlichen Verbinden-Trennen-Schleife. Die Datei „runtime.log“ enthält eine fortlaufende Reihe von Verbinden- und Trennen-Einträgen.	1345
Fehler: Lambda-Container kann nicht gestartet werden. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \"proc\" to rootfs \"\" 1346	1346
[FEHLER] -Laufzeit-Ausführungsfehler: Der Lambda-Container konnte nicht gestartet werden. <ggc-path>{"errorString": „Container-Mounts konnten nicht initialisiert werden: Fehler beim Maskieren der Greengrass-Wurzel im oberen Overlay-Verzeichnis: Fehler beim Erstellen des Maskengeräts im Verzeichnis: Datei existiert"} 1347	1347
[FEHLER] — Die Bereitstellung ist fehlgeschlagen. {"deploymentId": "<deployment-id>„, „errorString“: „Der Container-Testprozess mit der PID ist <pid>fehlgeschlagen: Status des Container-Prozesses: Exit-Status 1"} 1348	1348
Fehler: [FEHLER] — Laufzeit-Ausführungsfehler: Der Lambda-Container konnte nicht gestartet werden. <ggc-version><ggc-version>{"errorString": „Container-Mounts konnten nicht initialisiert werden: Fehler beim Erstellen von Overlay fs für Container: Mount-Overlay unter/////greengrass/ggc/packages<ggc-version><ggc-version><ggc-version>/dns:/, rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages upperdir=///rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages//rootfs/work\": zu viele Ebenen symbolischer Links greengrass/ggc/packages \"} rootfs/upper,workdir=/greengrass/ggc/packages 1349	1349
Fehler: [DEBUG] – Routen konnten nicht abgerufen werden. Nachricht wird verworfen. 1350	1350
Fehler: [Errno 24] Zu viele geöffnete Dateien <lambda-function> [Errno 24] Zu viele geöffnete Dateien 1350	1350
Fehler: Der DS-Server konnte nicht mit der Überwachung von Socket beginnen: listen unix<ggc-path>/ggc/socket/greengrass_ipc.sock: bind: ungültiges Argument 1350	1350
[INFO] (Kopierer) aws.greengrass. StreamManager: stdout. Verursacht durch: com.fasterxml.jackson.databind. JsonMappingException: Instant überschreitet den Mindest- oder Maximalwert des Sofortwerts 1351	1351
GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key 1351	1351
Bereitstellungsprobleme 1352	1352
Ihre aktuelle Bereitstellung funktioniert nicht und Sie möchten zu einer früheren, funktionierenden Bereitstellung zurückkehren. 1353	1353
Für die Bereitstellung wird der Fehler „403 Forbidden“ in den Protokollen angezeigt. 1356	1356

Ein ConcurrentDeployment Fehler tritt auf, wenn Sie den Befehl create-deployment zum ersten Mal ausführen.	1356
Fehler: Greengrass ist nicht zur Übernahme der Servicerolle berechtigt, die mit diesem Konto verknüpft ist; oder der Fehler: Fehlgeschlagen: TES-Servicerolle ist nicht mit diesem Konto verknüpft.	1068
Fehler: Download-Schritt in der Bereitstellung kann nicht ausgeführt werden. Fehler beim Herunterladen: Fehler beim Herunterladen der Gruppendefinitionsdatei:... x509: Zertifikat ist abgelaufen oder noch nicht gültig	1356
Die Bereitstellung wird nicht abgeschlossen.	1357
Fehler: Die ausführbaren Java- oder Java8-Dateien konnten nicht gefunden werden, oder der Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen. Fehler: Der Worker konnte nicht mit einem <worker-id>bestimmten Grund initialisiert werden. Die installierte Java-Version muss größer oder gleich 8 sein	1358
Die Bereitstellung wird nicht fertiggestellt und „runtime.log“ enthält mehrere Einträge für „1 Sekunde auf Anhalten des Containers gewartet“.	1358
Die Bereitstellung wird nicht abgeschlossen und runtime.log enthält "[FEHLER]-Greengrass-Bereitstellungsfehler: Fehler beim Melden des Bereitstellungsstatus an die Cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"	1359
<path>Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen Fehler: Fehler bei der Verarbeitung. Die Gruppenkonfiguration ist ungültig: 112 oder [119 0] haben keine RW-Rechte für die Datei:	1360
Fehler: < list-of-function-arns > sind für die Ausführung als Root konfiguriert, aber Greengrass ist nicht für die Ausführung von Lambda-Funktionen mit Root-Rechten konfiguriert.	1360
Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen Fehler: Greengrass-Bereitstellungsfehler: Download-Schritt in der Bereitstellung konnte nicht ausgeführt werden. Fehler bei der Verarbeitung: Die heruntergeladene Gruppendatei konnte nicht geladen werden: Die UID konnte nicht anhand des Benutzernamens gefunden werden, userName: ggc_user: user: unknown user ggc_user.	1361
Fehler: [FEHLER]-Laufzeitausführungsfehler: Lambda-Container kann nicht gestartet werden. {"errorString": "Fehler beim Initialisieren von Container-Mounts: Fehler beim	

Maskieren des Greengrass-Stamms im Overlay-Oberverzeichnis: Fehler beim Erstellen eines Maskengeräts im Verzeichnis <ggc-path>: Datei vorhanden"}	1361
Fehler: Bereitstellung <deployment-id>des Typs NewDeployment für Gruppe ist <group-id>fehlgeschlagen Fehler: Prozessstart fehlgeschlagen: container_linux.go:259: Das Starten des Container-Prozesses verursachte „process_linux.go:250: running exec setns process for init caused\" wait: no child processes\" ""	1362
Fehler: [WARN<host-prefix>] -MQTT [Client] Wählen Sie TCP: Lookup -ats.iot. <region>.amazonaws.com: kein solcher Host... [FEHLER] -Greengrass-Bereitstellungsfehler: Der Bereitstellungsstatus konnte nicht an die Cloud zurückgemeldet werden... net/http: Die Anfrage wurde beim Warten auf die Verbindung abgebrochen (Client.Timeout wurde beim Warten auf Header überschritten)	1362
Probleme beim Erstellen der Gruppe oder Funktion	1363
Fehler: Ihre Konfiguration 'IsolationMode' für die Gruppe ist ungültig.	1363
Fehler: Ihre Konfiguration IsolationMode " für die Funktion mit arn <function-arn>ist ungültig.	1364
Fehler: Die MemorySize Konfiguration für eine Funktion mit arn <function-arn>ist in IsolationMode = nicht erlaubtNoContainer.	1364
Fehler: Der Zugriff auf die Sysfs-Konfiguration für eine Funktion mit arn <function-arn>ist in = nicht zulässig. IsolationMode NoContainer	1364
Fehler: Die MemorySize Konfiguration für die Funktion mit arn <function-arn>ist in IsolationMode = GreengrassContainer erforderlich.	1364
Fehler: Die Funktion <function-arn>bezieht sich auf eine Ressource des Typs<resource-type>, der in IsolationMode = NoContainer nicht zulässig ist.	1365
Fehler: Die Ausführungskonfiguration für die Funktion mit dem ARN <function-arn> ist nicht zulässig.	1365
Erkennungsprobleme	1365
Fehler: Gerät gehört zu vielen Gruppen an; Geräte dürfen nicht mehr als 10 Gruppen angehören	1366
Probleme mit Machine Learning-Ressourcen	1366
Ungültiger MLModel Besitzer — GroupOwnerSetting wird in der ML-Modellressource angegeben, GroupPermission ist aber GroupOwner nicht vorhanden	445
NoContainer Die Funktion kann beim Anhängen von Ressourcen für Machine Learning keine Berechtigungen konfigurieren. <function-arn>bezieht sich auf eine Ressource für maschinelles Lernen <resource-id>mit der entsprechenden Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.	445

Die Funktion <function-arn>bezieht sich auf Machine Learning Learning-Ressource, für die <resource-id> ResourceAccessPolicy sowohl die Berechtigungen als auch die Ressource fehlen OwnerSetting.	446
Die Funktion <function-arn>bezieht sich auf die Machine Learning Learning-Ressource <resource-id>mit der Berechtigung\ "rw\", während die Einstellung für den Ressourcenbesitzer GroupPermission nur\ "ro\" zulässt.	446
NoContainer Die Funktion <function-arn>bezieht sich auf Ressourcen des verschachtelten Zielpfads.	446
Lambda <function-arn> erhält Zugriff auf die Ressource <resource-id>, indem dieselbe Gruppenbesitzer-ID freigegeben wird	446
AWS IoT Greengrass Kernprobleme bei Docker	1368
Fehler: Unbekannte Optionen: -no-include-email.	401
Warnung: IPv4 ist deaktiviert. Das Netzwerk wird nicht funktionieren.	402
Fehler: Eine Firewall blockiert die Freigabe von Dateien zwischen Fenstern und den Containern.	402
Fehler: Beim Aufrufen der GetAuthorizationToken Operation ist ein Fehler aufgetreten (AccessDeniedException): User: arn:aws:iam: :user/ <account-id>ist <user-name>nicht berechtigt,; ecr: on resource: * auszuführen GetAuthorizationToken	402
Fehler: Container kann für den Greengrass-Service nicht erstellt werden: Konflikt. Der Containername „/aws-iot-greengrass“ wird bereits verwendet.	1370
Fehler: [FATAL] -Fehler beim Zurücksetzen des Mount-Namespaces des Threads aufgrund eines unerwarteten Fehlers: „Vorgang nicht zulässig“. Zur Sicherstellung der Konsistenz wird GGC abstürzen und manuell neu gestartet werden.	1371
Fehlerbehebung mit Protokollen	1371
Fehlerbehebung bei Speicherproblemen	1373
Fehlerbehebung für Nachrichten	1373
Beheben von Timeout-Problemen während der Schattensynchronisierung	1374
Überprüfen Sie re:POST AWS	1375
Dokumentverlauf	1376
Frühere Aktualisierungen	1402

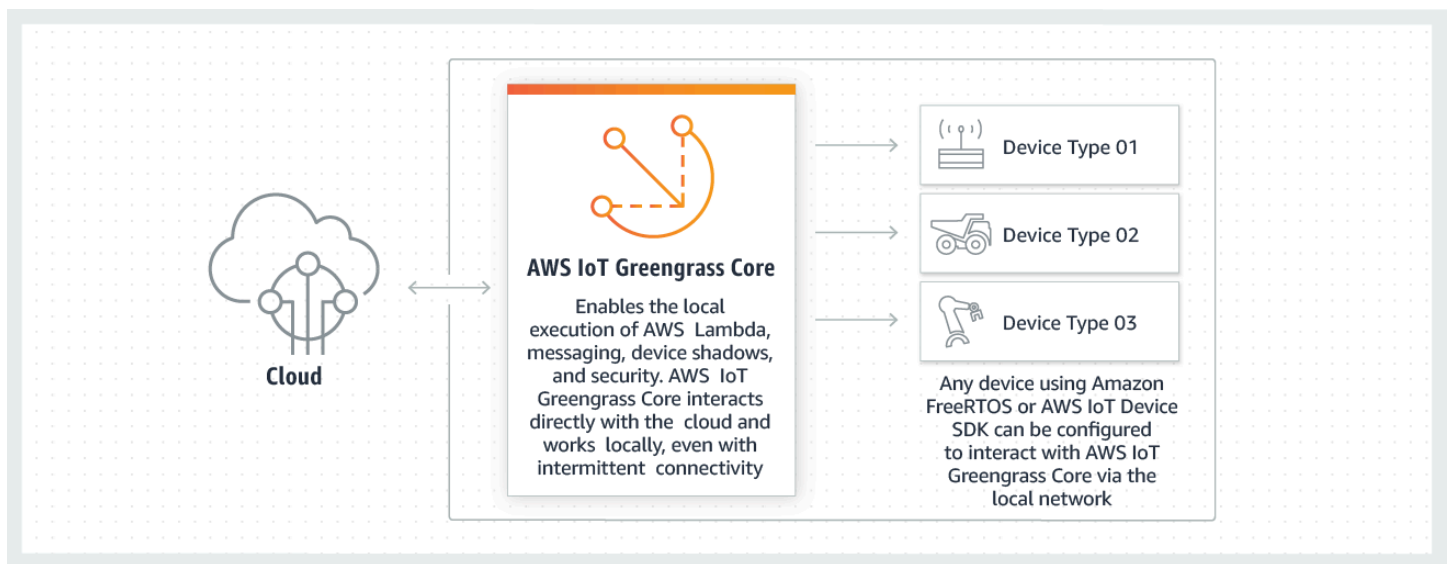
Hinweis zum Ende des Supports: Am 7. Oktober 2026 AWS wird der Support für eingestellt. AWS IoT Greengrass Version 1 Nach dem 7. Oktober 2026 können Sie nicht mehr auf die Ressourcen zugreifen. AWS IoT Greengrass V1 Weitere Informationen finden Sie unter [Migrieren von AWS IoT Greengrass Version 1](#).

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.

Was ist AWS IoT Greengrass?

AWS IoT Greengrass ist eine Software, die Cloud-Funktionen auf lokale Geräte erweitert. Dies ermöglicht es Geräten, Daten näher an der Informationsquelle zu erfassen und zu analysieren, selbstständig auf lokale Ereignisse zu reagieren und in lokalen Netzwerken sicher untereinander zu kommunizieren. Lokale Geräte können auch sicher mit dem kommunizieren AWS IoT Core und IoT-Daten dorthin exportieren AWS Cloud. AWS IoT Greengrass Entwickler können AWS Lambda Funktionen und vorgefertigte [Konnektoren](#) verwenden, um serverlose Anwendungen zu erstellen, die auf Geräten zur lokalen Ausführung bereitgestellt werden.

Das folgende Diagramm zeigt die grundlegende Architektur von AWS IoT Greengrass



AWS IoT Greengrass ermöglicht es Kunden, IoT-Geräte und Anwendungslogik zu erstellen. Insbesondere AWS IoT Greengrass bietet es eine cloudbasierte Verwaltung der Anwendungslogik, die auf Geräten ausgeführt wird. Lokal bereitgestellte Lambda-Funktionen und -Konnektoren werden durch lokale Ereignisse, Nachrichten aus der Cloud oder anderen Quellen ausgelöst.

AWS IoT Greengrass In kommunizieren Geräte sicher in einem lokalen Netzwerk und tauschen Nachrichten miteinander aus, ohne eine Verbindung zur Cloud herstellen zu müssen. AWS IoT Greengrass bietet einen lokalen pub/sub Nachrichtenmanager, der Nachrichten intelligent zwischenspeichern kann, falls die Konnektivität unterbrochen wird, sodass eingehende und ausgehende Nachrichten an die Cloud erhalten bleiben.

AWS IoT Greengrass schützt Benutzerdaten:

- Durch die sichere Authentifizierung und Autorisierung von Geräten.

- Durch sichere Verbindungen im lokalen Netzwerk.
- Zwischen lokalen Geräte und der Cloud.

Sicherheitsanmeldeinformationen für Geräte funktionieren innerhalb einer Gruppe, bis sie aufgehoben werden, auch wenn die Konnektivität mit der Cloud unterbrochen wird, sodass die Geräte weiterhin sicher lokal kommunizieren können.

AWS IoT Greengrass bietet sichere over-the-air Updates von Lambda-Funktionen.

AWS IoT Greengrass besteht aus:

- Software-Verteilungen
 - AWS IoT Greengrass Kernsoftware
 - AWS IoT Greengrass Kern-SDK
- Cloud-Service
 - AWS IoT Greengrass API
- Features
 - Lambda-Laufzeit
 - Shadows-Implementierung
 - Nachrichtenmanager
 - Gruppenmanagement
 - Discovery Service
 - Over-the-air Agent aktualisieren
 - Stream-Manager
 - Zugriff auf lokale Ressourcen
 - Lokale Machine Learning-Inferenz
 - Manager lokaler Secrets
 - Verbindungen mit werkseitiger Integration mit Services, Protokollen und Software

Themen

- [AWS IoT Greengrass Kernsoftware](#)
- [AWS IoT Greengrass Gruppen](#)
- [Geräte in AWS IoT Greengrass](#)

- [SDKs](#)
- [Unterstützte Plattformen und Anforderungen](#)
- [AWS IoT Greengrass lädt herunter](#)
- [Bitte geben Sie uns Feedback](#)
- [Installieren Sie die AWS IoT Greengrass Core-Software](#)
- [Den AWS IoT Greengrass Kern konfigurieren](#)

AWS IoT Greengrass Kernsoftware

Die AWS IoT Greengrass Core-Software bietet die folgenden Funktionen:

- Bereitstellung und lokale Ausführung von Konnektoren und Lambda-Funktionen.
- Verarbeiten Sie Datenströme lokal mit automatischen Exporten in die AWS Cloud.
- MQTT-Messaging über das lokale Netzwerk zwischen Geräten, Konnektoren und Lambda-Funktionen mithilfe verwalteter Abonnements.
- MQTT-Messaging zwischen AWS IoT Geräten, Konnektoren und Lambda-Funktionen mithilfe verwalteter Abonnements.
- Sichere Verbindungen zwischen Geräten und AWS Cloud Nutzung der Geräteauthentifizierung und -autorisierung.
- Lokale Shadow-Synchronisierung von Geräten. Shadows können so konfiguriert werden, dass sie mit dem synchronisiert werden AWS Cloud.
- Kontrollierter Zugriff auf lokale Geräte- und Volume-Ressourcen.
- Bereitstellung von Modellen für das maschinelle Lernen, die in der Cloud geschult werden, für die Ausführung lokaler Inferenzen.
- Automatische Erkennung von IP-Adressen, die Geräte in die Lage versetzt, das Greengrass Core-Gerät zu erkennen.
- Zentrale Bereitstellung neuer oder aktualisierter Gruppenkonfigurationen. Nach dem Download der Konfigurationsdaten wird das Core-Gerät automatisch neu gestartet.
- Sichere over-the-air (OTA) Softwareupdates von benutzerdefinierten Lambda-Funktionen.
- Sichere, verschlüsselte Speicherung lokaler Geheimnisse und kontrollierter Zugriff durch Konnektoren und Lambda-Funktionen.

AWS IoT Greengrass Kerninstanzen werden so konfiguriert, AWS IoT Greengrass APIs dass sie in der Cloud gespeicherte AWS IoT Greengrass Gruppensegmentdefinitionen erstellen und aktualisieren.

AWS IoT Greengrass Kern-Softwareversionen

AWS IoT Greengrass bietet mehrere Optionen für die Installation der AWS IoT Greengrass Core-Software, darunter Download-Dateien für tar.gz, ein Schnellstart-Skript und apt Installationen auf unterstützten Debian-Plattformen. Weitere Informationen finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

Die folgenden Reiter beschreiben, was in den AWS IoT Greengrass Core-Softwareversionen neu und geändert ist.

GGC v1.11

1.11.6

Fehlerbehebungen und Verbesserungen:

- Verbesserte Widerstandsfähigkeit bei plötzlichem Stromausfall während eines Einsatzes.
- Es wurde ein Problem behoben, bei dem die Beschädigung von Stream Manager-Daten dazu führen konnte, dass die AWS IoT Greengrass Core-Software nicht gestartet werden konnte.
- Es wurde ein Problem behoben, bei dem neue Client-Geräte in bestimmten Szenarien keine Verbindung zum Core herstellen konnten.
- Es wurde ein Problem behoben, bei dem Stream-Manager-Streamnamen nicht enthalten konnten .log.

1.11.5

Fehlerbehebungen und Verbesserungen:

- Allgemeine Leistungssteigerungen und Bugfixes.

1.11.4

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem mit dem Stream Manager behoben, das Upgrades auf die AWS IoT Greengrass Core-Software v1.11.3 verhinderte. Wenn Sie den Stream Manager verwenden, um Daten in die Cloud zu exportieren, können Sie jetzt ein OTA-Update verwenden, um eine frühere v1.x-Version der AWS IoT Greengrass Core-Software auf v1.11.4 zu aktualisieren.

- Allgemeine Leistungssteigerungen und Bugfixes.

1.11.3

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem behoben, das dazu führte, dass die AWS IoT Greengrass Core-Software, die auf einem Ubuntu-Gerät im Handumdrehen ausgeführt wurde, nach einem plötzlichen Stromausfall des Geräts nicht mehr reagierte.
- Es wurde ein Problem behoben, das zu einer verzögerten Übermittlung von MQTT-Nachrichten an langlebige Lambda-Funktionen führte.
- Es wurde ein Problem behoben, das dazu führte, dass MQTT-Nachrichten nicht korrekt gesendet wurden, wenn der `maxWorkItemCount` Wert auf einen Wert größer als gesetzt wurde. 1024
- Es wurde ein Problem behoben, das dazu führte, dass der OTA-Update-Agent den in der `keepAlive` Eigenschaft in angegebenen `KeepAlive` MQTT-Zeitraum ignorierte. [config.json](#)
- Allgemeine Leistungssteigerungen und Bugfixes.

Important

Wenn Sie Stream Manager verwenden, um Daten in die Cloud zu exportieren, führen Sie kein Upgrade von einer früheren Version AWS IoT Greengrass v1.x auf die Core-Software v1.11.3 durch. Wenn Sie den Stream Manager zum ersten Mal aktivieren, empfehlen wir dringend, zuerst die neueste Version der AWS IoT Greengrass Core-Software zu installieren.

1.11.1

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem behoben, das zu einem erhöhten Speicherverbrauch für den Stream-Manager führte.
- Es wurde ein Problem behoben, das dazu führte, dass der Stream-Manager die Sequenznummer des Streams auf zurücksetzte, 0 wenn das Greengrass-Core-Gerät länger als den angegebenen Zeitraum `time-to-live` (TTL) der Stream-Daten ausgeschaltet war.
- Es wurde ein Problem behoben, das verhinderte, dass der Stream Manager Wiederholungsversuche, Daten in den zu exportieren, korrekt unterbrach. AWS Cloud

1.11.0

Neue Funktionen:

- Ein Telemetrieagent auf dem Greengrass-Core sammelt lokale Telemetriedaten und veröffentlicht sie an AWS Cloud. Um die Telemetriedaten für die weitere Verarbeitung abzurufen, können Kunden eine EventBridge Amazon-Regel erstellen und ein Ziel abonnieren. Weitere Informationen finden Sie unter [Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten](#).
- Eine lokale HTTP-API gibt eine Momentaufnahme des aktuellen Status der lokalen Arbeitsprozesse zurück, die von gestartet wurden AWS IoT Greengrass. Weitere Informationen finden Sie unter [Aufrufen der lokalen Health Check-API](#).
- Ein [Stream-Manager](#) exportiert automatisch Daten nach Amazon S3 und AWS IoT SiteWise.

Mit neuen [Stream-Manager-Parametern](#) können Sie bestehende Streams aktualisieren und den Datenexport anhalten oder fortsetzen.

- Support für die Ausführung von Python 3.8.x Lambda-Funktionen auf dem Kern.
- Eine neue ggDaemonPort Eigenschaft [config.json](#), die verwendet wird, um die Greengrass-Core-IPC-Portnummer zu konfigurieren. Die Standard-Portnummer ist 8000.

Eine neue systemComponentAuthTimeout Eigenschaft, mit der Sie [config.json](#) das Timeout für die Greengrass-Core-IPC-Authentifizierung konfigurieren. Das Standard-Timeout beträgt 5000 Millisekunden.

- Die maximale Anzahl von AWS IoT Geräten pro AWS IoT Greengrass Gruppe wurde von 200 auf 2500 erhöht.

Die maximale Anzahl von Abonnements pro Gruppe wurde von 1000 auf 10000 erhöht.

Weitere Informationen finden Sie unter [AWS IoT Greengrass -Endpunkte und -Kontingente](#).

Fehlerbehebungen und Verbesserungen:

- Allgemeine Optimierung, die die Speicherauslastung der Greengrass-Serviceprozesse reduzieren kann.
- Ein neuer Laufzeitkonfigurationsparameter (mountAllBlockDevices) ermöglicht es Greengrass, Bind-Mounts zu verwenden, um alle Blockgeräte nach der Einrichtung von OverlayFS in einen Container zu mounten. Mit dieser Funktion wurde ein Problem behoben, das dazu führte, dass die Greengrass-Bereitstellung fehlschlug, wenn es sich /usr nicht in der / Hierarchie befindet.

- Es wurde ein Problem behoben, das zu einem AWS IoT Greengrass Kernausfall führte, wenn /tmp es sich um einen Symlink handelt.
- Es wurde ein Problem behoben, durch das der Greengrass Deployment Agent ungenutzte Modellartefakte für maschinelles Lernen aus dem `mlmodel_public` Ordner entfernen konnte.
- Allgemeine Leistungssteigerungen und Bugfixes.

Extended life versions

1.10.5

Fehlerbehebungen und Verbesserungen:

- Allgemeine Leistungssteigerungen und Bugfixes.

1.10.4

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem behoben, das dazu führte, dass die AWS IoT Greengrass Core-Software, die auf einem Ubuntu-Gerät im Handumdrehen ausgeführt wurde, nach einem plötzlichen Stromausfall des Geräts nicht mehr reagierte.
- Es wurde ein Problem behoben, das zu einer verzögerten Übermittlung von MQTT-Nachrichten an langlebige Lambda-Funktionen führte.
- Es wurde ein Problem behoben, das dazu führte, dass MQTT-Nachrichten nicht korrekt gesendet wurden, wenn der `maxWorkItemCount` Wert auf einen Wert größer als gesetzt wurde. 1024
- Es wurde ein Problem behoben, das dazu führte, dass der OTA-Update-Agent den in der `keepAlive` Eigenschaft in angegebenen KeepAlive MQTT-Zeitraum ignorierte. [config.json](#)
- Allgemeine Leistungssteigerungen und Bugfixes.

1.10.3

Fehlerbehebungen und Verbesserungen:

- Eine neue `systemComponentAuthTimeout` Eigenschaft, mit der Sie [config.json](#) das Timeout für die Greengrass-Core-IPC-Authentifizierung konfigurieren. Das Standard-Timeout beträgt 5000 Millisekunden.

- Es wurde ein Problem behoben, das zu einem erhöhten Speicherverbrauch für den Stream-Manager führte.

1.10.2

Fehlerbehebungen und Verbesserungen:

- Eine neue `mqttOperationTimeout` Eigenschaft in [config.json](#), mit der Sie das Timeout für Veröffentlichungs-, Abonnier- und Abmeldevorgänge in MQTT-Verbindungen festlegen.
- AWS IoT Core
- Allgemeine Leistungssteigerungen und Bugfixes.

1.10.1

Fehlerbehebungen und Verbesserungen:

- [Stream-Manager](#) ist widerstandsfähiger gegen Dateidatenbeschädigung.
- Es wurde ein Problem behoben, durch das auf Geräten mit Linux-Kernel 5.1 und höher ein Sysfs-Mount-Fehler verursacht wurde.
- Allgemeine Leistungssteigerungen und Bugfixes.

1.10.0

Neue Funktionen:

- Ein Stream-Manager, der Datenströme lokal verarbeitet und sie AWS Cloud automatisch in den exportiert. Diese Funktion erfordert Java 8 auf dem Greengrass-Core-Gerät. Weitere Informationen finden Sie unter [Verwalten von Daten-Streams](#).
- Ein neuer Greengrass-Docker-Anwendungsbereitstellungs-Konnektor, der eine Docker-Anwendung auf einem Core-Gerät ausführt. Weitere Informationen finden Sie unter [the section called “Docker-Anwendungsbereitstellung”](#).
- Ein neuer SiteWise IoT-Connector, der industrielle Gerätedaten von OPC-UA-Servern an Anlagenimmobilien in sendet. AWS IoT SiteWise Weitere Informationen finden Sie unter [the section called “IoT SiteWise”](#).
- Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, können auf Ressourcen für maschinelles Lernen in der Greengrass-Gruppe zugreifen. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Machine Learning-Ressourcen”](#).
- Support für persistente MQTT-Sitzungen mit AWS IoT. Weitere Informationen finden Sie unter [the section called “Persistente MQTT-Sitzungen mit AWS IoT Core”](#).

- Lokaler MQTT-Datenverkehr kann über einen anderen Port als den Standard-Port 8883 übertragen. Weitere Informationen finden Sie unter [the section called “MQTT-Port für lokales Messaging”](#).
- Neue queueFullPolicy Optionen im [AWS IoT Greengrass Core SDK](#) für die zuverlässige Veröffentlichung von Nachrichten über Lambda-Funktionen.
- Support für die Ausführung von Node.js 12.x Lambda-Funktionen auf dem Kern.
- Over-the-air (OTA) -Updates mit Hardware-Sicherheitsintegration können mit OpenSSL 1.1 konfiguriert werden.
- Allgemeine Leistungssteigerungen und Bugfixes.

1.9.4

Fehlerbehebungen und Verbesserungen:

- Allgemeine Leistungssteigerungen und Bugfixes.

1.9.3

Neue Funktionen:

- Support für ARMv6L. AWS IoT Greengrass Die Kernsoftware v1.9.3 oder höher kann auf Raspbian-Distributionen auf ARMv6L-Architekturen (z. B. auf Raspberry Pi Zero-Geräten) installiert werden.
- OTA-Updates auf Port 443 mit ALPN. Greengrass-Kerne, die Port 443 für MQTT-Verkehr verwenden, unterstützen jetzt over-the-air (OTA) -Softwareupdates. AWS IoT Greengrass verwendet die TLS-Erweiterung Application Layer Protocol Network (ALPN), um diese Verbindungen zu aktivieren. Weitere Informationen erhalten Sie unter [OTA-Updates der AWS IoT Greengrass Core-Software](#) und [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).

Fehlerbehebungen und Verbesserungen:

- Behebt einen in Version 1.9.0 eingeführten Fehler, der verhinderte, dass Lambda-Funktionen von Python 2.7 binäre Nutzlasten an andere Lambda-Funktionen senden konnten.
- Allgemeine Leistungssteigerungen und Bugfixes.

1.9.2

Neue Funktionen:

- Support für [OpenWrt](#). AWS IoT Greengrass Die Kernsoftware v1.9.2 oder höher kann auf OpenWrt Distributionen mit Armv8 (AArch64) - und ARMv7L-Architekturen installiert werden. Unterstützt derzeit keine ML-Inferenz. OpenWrt

1.9.1

Fehlerbehebungen und Verbesserungen:

- Es wird ein in v1.9.0 eingeführter Fehler behoben, der Nachrichten aus c1oud entfernt, die im Thema Platzhalterzeichen enthalten.

1.9.0

Neue Funktionen:

- Support für Lambda-Laufzeiten von Python 3.7 und Node.js 8.10. Lambda-Funktionen, die die Laufzeiten Python 3.7 und Node.js 8.10 verwenden, können jetzt auf einem Core ausgeführt werden. AWS IoT Greengrass (unterstützt AWS IoT Greengrass weiterhin die Laufzeiten Python 2.7 und Node.js 6.10.)
- Optimierte MQTT-Verbindungen. Der Greengrass Core baut weniger Verbindungen mit dem AWS IoT Core auf. Diese Änderung kann Betriebskosten hinsichtlich Gebühren, die auf der Anzahl der Verbindungen basieren, senken.
- Elliptic Curve (EC)-Schlüssel für den lokalen MQTT-Server. Der lokale MQTT-Server unterstützt EC-Schlüssel zusätzlich zu den RSA-Schlüsseln. (Das MQTT-Serverzertifikat verfügt über eine SHA-256-RSA-Signatur, unabhängig vom Schlüsseltyp.) Weitere Informationen finden Sie unter [the section called "Sicherheitsprinzipale"](#).

Fehlerbehebungen und Verbesserungen:

- Allgemeine Leistungssteigerungen und Bugfixes.

1.8.4

Es wurde ein Problem mit der Schattensynchronisierung und der erneuten Verbindung des Gerätezertifikatmanagers behoben.

Allgemeine Leistungssteigerungen und Bugfixes.

1.8.3

Allgemeine Leistungssteigerungen und Bugfixes.

1.8.2

Allgemeine Leistungssteigerungen und Bugfixes.

1.8.1

Allgemeine Leistungssteigerungen und Bugfixes.

1.8.0

Neue Funktionen:

- Konfigurierbare Standardzugriffsidentität für Lambda-Funktionen in der Gruppe. Diese Einstellung auf Gruppenebene bestimmt die Standardberechtigungen, die zum Ausführen von Lambda-Funktionen verwendet werden. Sie können die Benutzer-ID, Gruppen-ID oder beides festlegen. Einzelne Lambda-Funktionen können die Standardzugriffsidentität ihrer Gruppe überschreiben. Weitere Informationen finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).
- HTTPS-Datenverkehr über Port 443. HTTPS-Kommunikation kann so konfiguriert werden, dass sie über Port 443 anstatt über Standard-Port 8443 ausgeführt wird. Dies ergänzt die AWS IoT Greengrass Unterstützung für die TLS-Erweiterung Application Layer Protocol Network (ALPN) und ermöglicht es dem gesamten Greengrass-Messaging-Verkehr — sowohl MQTT als auch HTTPS —, Port 443 zu verwenden. Weitere Informationen finden Sie unter [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).
- IDs AWS IoT Vorhersehbar benannter Client für Verbindungen. Diese Änderung ermöglicht die Unterstützung für AWS IoT Device Defender und [AWS IoT -Lebenszyklus-Ereignisse](#), damit Sie Benachrichtigungen für Verbindungs-, Abmelde-, Abonnement- und Abonnementbeendungsereignisse erhalten. Eine vorhersehbare Benennung erleichtert auch die Erstellung von Logik rund um die Verbindung IDs (z. B. die Erstellung von [Abonnementrichtlinienvorlagen](#) auf der Grundlage von Zertifikatsattributen). Weitere Informationen finden Sie unter [the section called “Client IDs für MQTT-Verbindungen mit AWS IoT”](#).

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem mit der Schattensynchronisierung und der erneuten Verbindung des Gerätezertifikatmanagers behoben.
- Allgemeine Leistungssteigerungen und Bugfixes.

1.7.1

Neue Funktionen:

- Greengrass-Konnektoren bieten eine integrierte Integration mit lokaler Infrastruktur, Geräteprotokollen und anderen Cloud-Diensten. AWS Weitere Informationen finden Sie unter [Integrieren von Services und Protokollen mit Konnektoren](#).
- AWS IoT Greengrass erstreckt sich AWS Secrets Manager auf Kerngeräte, wodurch Ihre Passwörter, Token und andere Geheimnisse für Konnektoren und Lambda-Funktionen verfügbar sind. Secrets werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).
- Unterstützung für eine Hardware-Root of Trust Sicherheitsoption. Weitere Informationen finden Sie unter [the section called “Integration von Hardware-Sicherheit”](#).
- Isolations- und Berechtigungseinstellungen, die es Lambda-Funktionen ermöglichen, ohne Greengrass-Container ausgeführt zu werden und die Berechtigungen eines bestimmten Benutzers und einer bestimmten Gruppe zu verwenden. Weitere Informationen finden Sie unter [the section called “Steuerung der Ausführung von Greengrass Lambda-Funktionen”](#).
- Sie können AWS IoT Greengrass in einem Docker-Container (unter Windows, macOS oder Linux) ausführen, indem Sie Ihre Greengrass-Gruppe so konfigurieren, dass sie ohne Containerisierung ausgeführt wird. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass In einem Docker-Container ausführen”](#).
- MQTT-Messaging auf Port 443 mit Application Layer Protocol Negotiation (ALPN) oder die Verbindung über einen Netzwerk-Proxy. Weitere Informationen finden Sie unter [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).
- Die SageMaker AI Neo Deep Learning-Laufzeit, die Modelle für maschinelles Lernen unterstützt, die vom SageMaker AI Neo Deep Learning-Compiler optimiert wurden. Weitere Informationen zur Neo Deep Learning-Laufzeit finden Sie unter [the section called “Laufzeiten und Bibliotheken für ML-Inferenz”](#).
- Unterstützung für Raspbian Stretch (2018-06-27) auf Raspberry Pi Core Geräten.

Fehlerbehebungen und Verbesserungen:

- Allgemeine Leistungssteigerungen und Bugfixes.

Darüber hinaus sind mit dieser Version die folgenden Funktionen verfügbar:

- Der AWS IoT Gerätetester AWS IoT Greengrass, mit dem Sie überprüfen können, ob Ihre CPU-Architektur, Kernel-Konfiguration und Treiber AWS IoT Greengrass funktionieren. Weitere Informationen finden Sie unter [Verwenden von AWS IoT Device Tester für AWS IoT Greengrass V1](#).

- Die AWS IoT Greengrass Core-Software-, AWS IoT Greengrass Core SDK- und AWS IoT Greengrass Machine Learning Learning-SDK-Pakete können über Amazon heruntergeladen werden CloudFront. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass lädt herunter”](#).

1.6.1

Neue Funktionen:

- Ausführbare Lambda-Dateien, die Binärcode auf dem Greengrass-Kern ausführen. Verwenden Sie das neue AWS IoT Greengrass Core SDK für C, um ausführbare Lambda-Dateien in C und C++ zu schreiben. Weitere Informationen finden Sie unter [the section called “Ausführbare Lambda-Dateien”](#).
- Optionaler Nachrichten-Cache im lokalen Speicher, der einen Neustart überdauert. Sie können die Speichereinstellungen für MQTT-Nachrichten konfigurieren, die in der Warteschlange auf die Verarbeitung warten. Weitere Informationen finden Sie unter [the section called “MQTT-Nachrichtenwarteschlange”](#).
- Das konfigurierbare maximale Verbindungswiederholungsintervall bei getrenntem Core-Gerät. Weitere Informationen finden Sie unter der `mqttMaxConnectionRetryInterval`-Eigenschaft in [the section called “AWS IoT Greengrass Kernkonfigurationsdatei”](#).
- Der lokale Ressourcenzugriff auf das `/proc`-Verzeichnis des Hosts. Weitere Informationen finden Sie unter [Zugreifen auf lokale Ressourcen](#).
- Konfigurierbares Schreibverzeichnis. Die AWS IoT Greengrass Core-Software kann an schreibgeschützten Speicherorten und an Speicherorten mit Lese-/Schreibzugriff bereitgestellt werden. Weitere Informationen finden Sie unter [the section called “Konfigurieren eines Schreibverzeichnisses”](#).

Fehlerbehebungen und Verbesserungen:

- Leistungsverbesserung zur Veröffentlichung von Nachrichten innerhalb des Greengrass-Cores und zwischen Geräten und dem Core.
- Die Rechenressourcen, die für die Verarbeitung von Protokollen erforderlich sind, die von benutzerdefinierten Lambda-Funktionen generiert wurden, wurden reduziert.

1.5.0

Neue Funktionen:

- AWS IoT Greengrass Machine Learning (ML) -Inferenz ist allgemein verfügbar. Sie können ML-Inferenz lokal auf AWS IoT Greengrass -Geräten mithilfe von Modellen durchführen,

die in der Cloud erstellt und trainiert werden. Weitere Informationen finden Sie unter [Durchführen von Machine Learning-Inferenzen](#).

- Die Lambda-Funktionen von Greengrass unterstützen jetzt zusätzlich zu JSON auch Binärdaten als Eingabe-Payload. Um diese Funktion nutzen zu können, müssen Sie auf AWS IoT Greengrass Core SDK Version 1.1.0 aktualisieren, die Sie von der [AWS IoT Greengrass Core](#) SDK-Downloadseite herunterladen können.

Fehlerbehebungen und Verbesserungen:

- Verringerung des allgemeinen Speicherbedarfs.
- Leistungsverbesserungen für das Senden von Nachrichten in die Cloud.
- Leistungs- und Stabilitätsverbesserungen für den Download-Agenten, Device-Certificate-Manager und OTA-Update-Agenten.
- Kleinere Fehlerbehebungen.

1.3.0

Neue Funktionen:

- Over-the-air (OTA) Update-Agent, der in der Cloud bereitgestellte Greengrass-Aktualisierungsaufträge verarbeiten kann. Den Agenten finden Sie im neuen /greengrass/ota-Verzeichnis. Weitere Informationen finden Sie unter [OTA-Updates der AWS IoT Greengrass Core-Software](#).
- Mit der Funktion für den Zugriff auf lokale Ressourcen können Greengrass Lambda-Funktionen auf lokale Ressourcen, wie Peripheriegeräte und Volumes, zugreifen. Weitere Informationen finden Sie unter [Greifen Sie mit Lambda-Funktionen und Konnektoren auf lokale Ressourcen zu](#).

1.1.0

Neue Funktionen:

- Bereitgestellte AWS IoT Greengrass Gruppen können zurückgesetzt werden, indem Lambda-Funktionen, Abonnements und Konfigurationen gelöscht werden. Weitere Informationen finden Sie unter [the section called “Zurücksetzen von Bereitstellungen”](#).
- Support für Node.js 6.10- und Java 8 Lambda-Laufzeiten zusätzlich zu Python 2.7.

Um von der vorherigen Version des Kerns zu migrieren: AWS IoT Greengrass

- Kopieren Sie die Zertifikate aus dem Ordner /greengrass/configuration/certs in den Ordner /greengrass/certs.

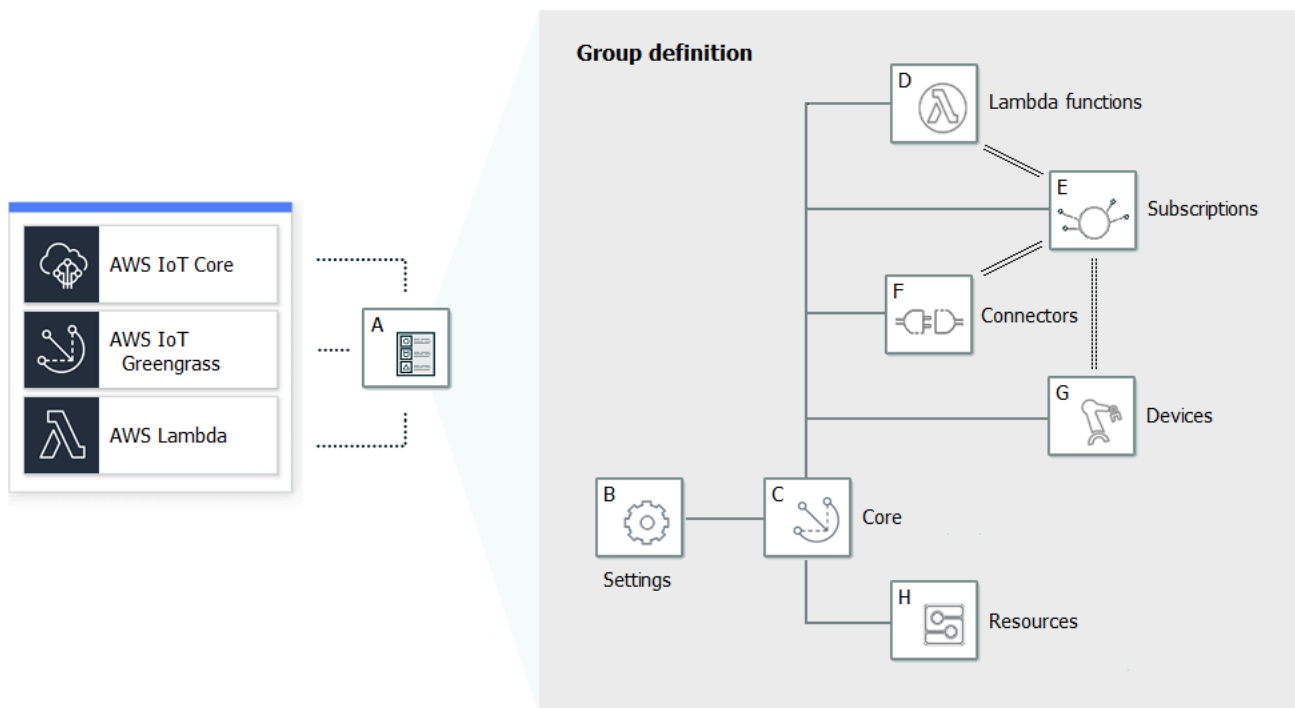
- Kopieren Sie `/greengrass/configuration/config.json` in `/greengrass/config/config.json`.
- Führen Sie `/greengrass/ggc/core/greengrassd` statt `/greengrass/greengrassd` aus.
- Stellen Sie die Gruppe auf dem neuen Core bereit.

1.0.0

Erste Version

AWS IoT Greengrass Gruppen

Eine Greengrass-Gruppe besteht aus einer Sammlung von Einstellungen und Komponenten, z. B. einem Greengrass-Kern, Geräten und Abonnements. Gruppen werden verwendet, um einen Interaktionsbereich definieren. Beispiel: Eine Gruppe kann ein Stockwerk eines Gebäudes, einen LKW oder eine ganze Mine repräsentieren. Das folgende Diagramm zeigt die Komponenten, aus denen eine Greengrass-Gruppe bestehen kann.



Im vorhergehenden Diagramm:

A: Definition der Greengrass-Gruppe

Informationen zu Gruppeneinstellungen und Komponenten.

B: Greengrass-Gruppeneinstellungen

Dazu zählen:

- Greengrass-Gruppenrolle.
- Zertifizierungsstelle und Konfiguration der lokalen Verbindung.
- Informationen zur Greengrass-Kernkonnektivität.
- Standard-Lambda-Laufzeitumgebung. Weitere Informationen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).
- CloudWatch und Konfiguration der lokalen Protokolle. Weitere Informationen finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

C: Greengrass-Kern

Das AWS IoT Ding (Gerät), das den Greengrass-Kern darstellt. Weitere Informationen finden Sie unter [the section called “Konfiguriere den AWS IoT Greengrass Kern”](#).

D: Definition der Lambda-Funktion

Eine Liste von Lambda-Funktionen, die lokal auf dem Core ausgeführt werden, mit zugehörigen Konfigurationsdaten. Weitere Informationen finden Sie unter [Lokale Lambda-Funktionen ausführen](#).

E: Abonnementdefinition

Eine Liste der Abonnements, die die Kommunikation mit MQTT-Nachrichten aktivieren. Mit einem Abonnement wird Folgendes definiert:

- Eine Nachrichtenquelle und ein Nachrichtenziel. Dies können Client-Geräte, Lambda-Funktionen AWS IoT Core, Konnektoren und der lokale Shadow-Service sein.
- Ein Thema oder Betreff, das zum Filtern von Nachrichten verwendet wird.

Weitere Informationen finden Sie unter [the section called “Verwaltete Abonnements im MQTT Messaging-Workflow”](#).

F: Konnektordefinition

Eine Liste von Konnektoren, die lokal auf dem Core ausgeführt werden, mit zugehörigen Konfigurationsdaten. Weitere Informationen finden Sie unter [Integrieren von Services und Protokollen mit Konnektoren](#).

G: Gerätedefinition

Eine Liste von AWS IoT Dingen (bekannt als Client-Geräte oder Geräte), die Mitglieder der Greengrass-Gruppe sind, mit zugehörigen Konfigurationsdaten. Weitere Informationen finden Sie unter [the section called “Geräte in AWS IoT Greengrass”](#).

H: Ressourcendefinition

Eine Liste der lokalen Ressourcen, Machine Learning-Ressourcen und geheimen Ressourcen auf dem Greengrass-Kern, mit den zugehörigen Konfigurationsdaten. Weitere Informationen finden Sie unter [Zugreifen auf lokale Ressourcen](#), [Durchführen von Machine Learning-Inferenzen](#) und [Bereitstellen von Secrets für den Core](#).

Bei der Bereitstellung werden die Greengrass-Gruppendefinition, die Lambda-Funktionen, Konnektoren, Ressourcen und die Abonnementtabelle auf das Kerngerät kopiert. Weitere Informationen finden Sie unter [Stellen Sie AWS IoT Greengrass Gruppen bereit](#).

Geräte in AWS IoT Greengrass

Eine Greengrass-Gruppe kann zwei AWS IoT Gerätetypen enthalten:

Greengrass-Kern

Ein Greengrass-Core ist ein Gerät, auf dem die AWS IoT Greengrass Core-Software ausgeführt wird, sodass es direkt mit dem Dienst AWS IoT Core und dem AWS IoT Greengrass Dienst kommunizieren kann. Ein Core verfügt über ein eigenes Gerätezertifikat, das für die Authentifizierung verwendet wird. AWS IoT Core Er hat einen Geräteshadow und einen Eintrag in der AWS IoT Core Registrierung. Greengrass-Cores führen eine lokale Lambda-Laufzeit, einen Deployment-Agenten und einen IP-Adress-Tracker aus, der IP-Adressinformationen an den AWS IoT Greengrass Service sendet, damit Client-Geräte ihre Gruppen- und Kernverbindungsinformationen automatisch erkennen können. Weitere Informationen finden Sie unter [the section called “Konfiguriere den AWS IoT Greengrass Kern”](#).

Note

Eine Greengrass-Gruppe muss genau einen Kern enthalten.

Client-Gerät

Client-Geräte (auch verbundene Geräte, Greengrass-Geräte oder Geräte genannt) sind Geräte, die über MQTT eine Verbindung zu einem Greengrass-Core herstellen. Sie verfügen über ein eigenes Gerätezertifikat für die AWS IoT Core Authentifizierung, einen Geräteshadow und einen Eintrag in der Registrierung. AWS IoT Core Client-Geräte können [FreeRTOS](#) ausführen oder das [AWS IoT Device SDK](#) oder die [AWS IoT Greengrass Discovery-API verwenden, um Discovery-Informationen](#) abzurufen, die für die Verbindung und Authentifizierung mit dem Core in derselben Greengrass-Gruppe verwendet werden. Informationen darüber, wie Sie mit der AWS IoT Konsole ein Client-Gerät für erstellen und konfigurieren, finden Sie unter [AWS IoT Greengrass the section called "Modul 4: Interaktion mit Client-Geräten in einer AWS IoT Greengrass Gruppe"](#) Beispiele, die Ihnen zeigen, wie Sie mit AWS CLI dem ein Client-Gerät erstellen und konfigurieren AWS IoT Greengrass, finden Sie [create-device-definition](#) in der AWS CLI Befehlsreferenz.

In einer Greengrass-Gruppe können Sie Abonnements erstellen, die es Client-Geräten ermöglichen, über MQTT mit Lambda-Funktionen, Connectors und anderen Client-Geräten in der Gruppe sowie mit AWS IoT Core oder dem lokalen Shadow-Service zu kommunizieren. MQTT-Nachrichten werden über den Kern geleitet. Wenn das Core-Gerät die Konnektivität zur Cloud verliert, können Client-Geräte weiterhin über das lokale Netzwerk kommunizieren. Client-Geräte können unterschiedlich groß sein, von kleineren Geräten auf Mikrocontroller-Basis bis hin zu großen Geräten. Derzeit kann eine Greengrass-Gruppe bis zu 2.500 Client-Geräte enthalten. Ein Client-Gerät kann Mitglied von bis zu 10 Gruppen sein.

Note

OPC-UA ist ein Standard für den Informationsaustausch für die industrielle Kommunikation. [Um die Unterstützung für OPC-UA auf dem Greengrass-Core zu implementieren, können Sie den IoT-Connector verwenden. SiteWise](#) Der Konnektor sendet Daten von Industriegeräten von OPC-UA-Servern an Anlagen in. AWS IoT SiteWise

Die folgende Tabelle zeigt, wie diese Gerätetypen verknüpft sind.

	Core	Device
Certificate	✓	✓
IoT Policy	✓	✓
IoT Thing	✓	✓
Device use	Gateway	Sensor and/or Actuator
Software	AWS IoT Greengrass Core Software	Amazon FreeRTOS / AWS IoT Device SDK
Group membership	✓	✓
Functions outside a Greengrass Group	✗	✓

Das AWS IoT Greengrass Kerngerät speichert Zertifikate an zwei Orten:

- Kern-Gerätezertifikat in `/greengrass-root/certs`. In der Regel wird das Kern-Gerätezertifikat als `hash.cert.pem` bezeichnet (z. B. `86c84488a5.cert.pem`). Dieses Zertifikat wird vom AWS IoT Client für die gegenseitige Authentifizierung verwendet, wenn der Core eine Verbindung zu den AWS IoT Core AWS IoT Greengrass AND-Diensten herstellt.
- MQTT-Serverzertifikat in `/greengrass-root/ggc/var/state/server`. Das MQTT-Serverzertifikat hat den Namen `server.crt`. Dieses Zertifikat wird für die gegenseitige Authentifizierung zwischen dem lokalen MQTT-Server (auf dem Greengrass-Kern) und Greengrass-Geräten verwendet.

 Note

greengrass-root stellt den Pfad dar, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

SDKs

Die folgenden AWS bereitgestellten Dateien SDKs werden für die Arbeit mit AWS IoT Greengrass verwendet:

AWS SDK

Verwenden Sie das AWS SDK, um Anwendungen zu erstellen, die mit beliebigen AWS Services interagieren, einschließlich Amazon S3, Amazon DynamoDB, AWS IoT AWS IoT Greengrass, und mehr. Im Zusammenhang mit können Sie das AWS SDK in bereitgestellten Lambda-Funktionen verwenden, um direkte Aufrufe an jeden AWS Dienst zu tätigen. AWS IoT Greengrass Weitere Informationen finden Sie unter [AWS SDKs](#).

 Note

Die für Greengrass spezifischen Operationen, die in der verfügbar sind, AWS SDKs sind auch in der [AWS IoT Greengrass API](#) und [AWS CLI](#) verfügbar.

AWS IoT Geräte-SDK

Das AWS IoT Geräte-SDK unterstützt Geräte dabei, sich mit AWS IoT Core und zu verbinden AWS IoT Greengrass. Weitere Informationen finden Sie unter [AWS IoT Gerät SDKs](#) im AWS IoT Entwicklerhandbuch.

Client-Geräte können jede der AWS IoT Device SDK v2-Plattformen verwenden, um Konnektivätsinformationen für einen Greengrass-Core zu ermitteln. Zu den Konnektivätsinformationen gehören:

- Die IDs der Greengrass-Gruppen, zu denen das Client-Gerät gehört.
- Die IP-Adressen des Greengrass-Kerns in jeder Gruppe. Diese werden auch als Kernendpunkte bezeichnet.

- Das Gruppen-CA-Zertifikat, das Geräte für die gegenseitige Authentifizierung mit dem Core verwenden. Weitere Informationen finden Sie unter [the section called “Geräteverbindung – Workflow”](#).

 Note

In Version 1 des AWS IoT Geräts SDKs bieten nur die C++- und Python-Plattformen integrierte Discovery-Unterstützung.

AWS IoT Greengrass Kern-SDK

Das AWS IoT Greengrass Core SDK ermöglicht es Lambda-Funktionen, mit dem Greengrass-Kern zu interagieren, Nachrichten zu veröffentlichen AWS IoT, mit dem lokalen Shadow-Service zu interagieren, andere bereitgestellte Lambda-Funktionen aufzurufen und auf geheime Ressourcen zuzugreifen. Dieses SDK wird von Lambda-Funktionen verwendet, die auf einem AWS IoT Greengrass Kern ausgeführt werden. Weitere Informationen finden Sie unter [AWS IoT Greengrass Kern-SDK](#).

AWS IoT Greengrass SDK für Machine Learning

Das AWS IoT Greengrass Machine Learning SDK ermöglicht es Lambda-Funktionen, Modelle für maschinelles Lernen zu nutzen, die auf dem Greengrass-Kern als Ressourcen für maschinelles Lernen bereitgestellt werden. Dieses SDK wird von Lambda-Funktionen verwendet, die auf einem AWS IoT Greengrass Core ausgeführt werden und mit einem lokalen Inferenzdienst interagieren. Weitere Informationen finden Sie unter [AWS IoT Greengrass SDK für Machine Learning](#).

Unterstützte Plattformen und Anforderungen

Auf den folgenden Registerkarten sind die unterstützten Plattformen und die Anforderungen für die AWS IoT Greengrass Core-Software aufgeführt.

 Note

Sie können die AWS IoT Greengrass Core-Software von den [AWS IoT Greengrass Core-Software-Downloads](#) herunterladen.

GGC v1.11

Unterstützte Plattformen:

- Architektur: Armv7l
 - Betriebssystem: Linux
 - Betriebssystem: Linux ([OpenWrt](#))
- Architektur: Armv8 () AArch64
 - Betriebssystem: Linux
 - Betriebssystem: Linux ([OpenWrt](#))
- Architektur: Armv6l
 - Betriebssystem: Linux
- Architektur: x86_64
 - Betriebssystem: Linux
- Windows-, MacOS- und Linux-Plattformen können AWS IoT Greengrass in einem Docker-Container ausgeführt werden. Weitere Informationen finden Sie unter [the section called “ AWS IoT Greengrass In einem Docker-Container ausführen”](#).

Voraussetzungen:

- Für die AWS IoT Greengrass Core-Software stehen mindestens 128 MB Festplattenspeicher zur Verfügung. Wenn Sie den [OTA-Update-Agent](#) verwenden, sind mindestens 400 MB erforderlich.
- Der AWS IoT Greengrass Core-Software sind mindestens 128 MB RAM zugewiesen. Bei aktiviertem [Stream-Manager](#) beträgt das Minimum 198 MB RAM.



Note

Stream Manager ist standardmäßig aktiviert, wenn Sie die Option Standardgruppenerstellung auf der AWS IoT Konsole verwenden, um Ihre Greengrass-Gruppe zu erstellen.

- Linux-Kernel-Version:
 - Die Linux-Kernel-Version 4.4 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass mit [Containern](#) zu unterstützen.

- Die Linux-Kernel-Version 3.17 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass ohne Container zu unterstützen. In dieser Konfiguration muss die standardmäßige Containerisierung der Lambda-Funktion für die Greengrass-Gruppe auf **Kein Container** gesetzt werden. Detaillierte Anweisungen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).
- [GNU C Library \(Glibc\)](#) Version 2.14 oder höher. OpenWrt Distributionen benötigen die [Musl C Library](#) Version 1.1.16 oder höher.
- Das Verzeichnis `/var/run` muss auf dem Gerät vorhanden sein.
- Die Dateien `/dev/stdin`, `/dev/stdout` und `/dev/stderr` müssen verfügbar sein.
- Der Hard- und Softlink-Schutz muss auf dem Gerät aktiviert sein. Andernfalls AWS IoT Greengrass kann es nur im unsicheren Modus mit dem Flag ausgeführt werden. `-i`
- Die folgenden Linux-Kernel-Konfigurationen müssen auf dem Gerät aktiviert sein:
 - Namespace
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`
 - `CONFIG_USER_NS`
 - `CONFIG_PID_NS`
 - Cgroups:
 - `CONFIG_CGROUP_DEVICE`
 - `CONFIG_CGROUPS`
 - `CONFIG_MEMCG`

Der Kernel muss [cgroups](#) unterstützen. Bei der Ausführung AWS IoT Greengrass mit [Containern](#) gelten die folgenden Anforderungen:

- Die Speicher-Cgroup muss aktiviert und gemountet sein, AWS IoT Greengrass damit das Speicherlimit für Lambda-Funktionen festgelegt werden kann.
- Die Geräte-Cgroup muss aktiviert und gemountet sein, wenn Lambda-Funktionen mit [lokalem Ressourcenzugriff zum](#) Öffnen von Dateien auf dem AWS IoT Greengrass Kerngerät verwendet werden.
- Weitere:
 - `CONFIG_POSIX_MQUEUE`
 - `CONFIG_OVERLAY_FS`
 - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`

- CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Das Stammzertifikat für Amazon S3 AWS IoT muss im System Trust Store vorhanden sein.
 - [Stream Manager](#) benötigt zusätzlich zu den Basisspeicheranforderungen der AWS IoT Greengrass Core-Software die Laufzeit von Java 8 und mindestens 70 MB RAM. Stream Manager ist standardmäßig aktiviert, wenn Sie die Option Standardgruppenerstellung auf der AWS IoT Konsole verwenden. Stream Manager wird auf OpenWrt Distributionen nicht unterstützt.
 - Bibliotheken, die die [AWS Lambda Laufzeit](#) unterstützen, die für die Lambda-Funktionen erforderlich ist, die Sie lokal ausführen möchten. Erforderliche Bibliotheken müssen auf dem Core installiert und zur PATH-Umgebungsvariablen hinzugefügt werden. Auf einem Core können mehrere Bibliotheken installiert werden.
 - [Python](#) Version 3.8 für Funktionen, die die Python 3.8-Laufzeit verwenden.
 - [Python](#) Version 3.7 für Funktionen, die die Laufzeit Python 3.7 verwenden.
 - [Python](#) Version 2.7 für Funktionen, die die Laufzeit Python 2.7 verwenden.
 - [Node.js](#)-Version 12.x für Funktionen, die die Node.js 12.x-Laufzeit verwenden.
 - [Java](#) Version 8 oder höher für Funktionen, die die Laufzeit Java 8 verwenden.

 Note

Das Ausführen von Java auf einer OpenWrt Distribution wird nicht offiziell unterstützt. Wenn Ihr OpenWrt Build jedoch Java-Unterstützung bietet, können Sie möglicherweise in Java verfasste Lambda-Funktionen auf Ihren Geräten ausführen. OpenWrt

Weitere Informationen zur AWS IoT Greengrass Unterstützung von Lambda-Laufzeiten finden Sie unter. [Lokale Lambda-Funktionen ausführen](#)

- Die folgenden Shell-Befehle (nicht die BusyBox Varianten) werden vom [over-the-air \(OTA-\) Update-Agent](#) benötigt:
 - wget
 - realpath

- tar
- readlink
- basename
- dirname
- pidof
- df
- grep
- umount
- mv
- gzip
- mkdir
- rm
- ln
- cut
- cat
- /bin/bash

GGC v1.10

Unterstützte Plattformen:

- Architektur: Armv7l
 - Betriebssystem: Linux
 - Betriebssystem: Linux ([OpenWrt](#))
- Architektur: Armv8 () AArch64
 - Betriebssystem: Linux
 - Betriebssystem: Linux ([OpenWrt](#))
- Architektur: Armv6l
 - Betriebssystem: Linux
- Architektur: x86_64

- Windows-, MacOS- und Linux-Plattformen können AWS IoT Greengrass in einem Docker-Container ausgeführt werden. Weitere Informationen finden Sie unter [the section called “ AWS IoT Greengrass In einem Docker-Container ausführen”](#).

Voraussetzungen:

- Für die AWS IoT Greengrass Core-Software stehen mindestens 128 MB Festplattenspeicher zur Verfügung. Wenn Sie den [OTA-Update-Agent](#) verwenden, sind mindestens 400 MB erforderlich.
- Der AWS IoT Greengrass Core-Software sind mindestens 128 MB RAM zugewiesen. Bei aktiviertem [Stream-Manager](#) beträgt das Minimum 198 MB RAM.



Note

Stream Manager ist standardmäßig aktiviert, wenn Sie die Option Standardgruppenerstellung auf der AWS IoT Konsole verwenden, um Ihre Greengrass-Gruppe zu erstellen.

- Linux-Kernel-Version:
 - Die Linux-Kernel-Version 4.4 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass mit [Containern](#) zu unterstützen.
 - Die Linux-Kernel-Version 3.17 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass ohne Container zu unterstützen. In dieser Konfiguration muss die standardmäßige Containerisierung der Lambda-Funktion für die Greengrass-Gruppe auf Kein Container gesetzt werden. Detaillierte Anweisungen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).
- [GNU C Library \(Glibc\)](#) Version 2.14 oder höher. OpenWrt Distributionen benötigen die [Musl C Library](#) Version 1.1.16 oder höher.
- Das Verzeichnis `/var/run` muss auf dem Gerät vorhanden sein.
- Die Dateien `/dev/stdin`, `/dev/stdout` und `/dev/stderr` müssen verfügbar sein.
- Der Hard- und Softlink-Schutz muss auf dem Gerät aktiviert sein. Andernfalls AWS IoT Greengrass kann es nur im unsicheren Modus mit dem Flag ausgeführt werden. `-i`
- Die folgenden Linux-Kernel-Konfigurationen müssen auf dem Gerät aktiviert sein:
 - Namespace
 - `CONFIG_IPC_NS`

- CONFIG_UTS_NS
- CONFIG_USER_NS
- CONFIG_PID_NS
- Cgroups:
 - CONFIG_CGROUP_DEVICE
 - CONFIG_CGROUPS
 - CONFIG_MEMCG

Der Kernel muss [cgroups](#) unterstützen. Bei der Ausführung AWS IoT Greengrass mit [Containern](#) gelten die folgenden Anforderungen:

- Die Speicher-Cgroup muss aktiviert und gemountet sein, AWS IoT Greengrass damit das Speicherlimit für Lambda-Funktionen festgelegt werden kann.
- Die Geräte-Cgroup muss aktiviert und gemountet sein, wenn Lambda-Funktionen mit [lokalem Ressourcenzugriff zum](#) Öffnen von Dateien auf dem AWS IoT Greengrass Kerngerät verwendet werden.
- Weitere:
 - CONFIG_POSIX_QUEUE
 - CONFIG_OVERLAY_FS
 - CONFIG_HAVE_ARCH_SECCOMP_FILTER
 - CONFIG_SECCOMP_FILTER
 - CONFIG_KEYS
 - CONFIG_SECCOMP
 - CONFIG_SHMEM
- Das Stammzertifikat für Amazon S3 AWS IoT muss im System Trust Store vorhanden sein.
- [Stream Manager](#) benötigt zusätzlich zu den Basisspeicheranforderungen der AWS IoT Greengrass Core-Software die Laufzeit von Java 8 und mindestens 70 MB RAM. Stream Manager ist standardmäßig aktiviert, wenn Sie die Option Standardgruppenerstellung auf der AWS IoT Konsole verwenden. Stream Manager wird auf OpenWrt Distributionen nicht unterstützt.
- Bibliotheken, die die [AWS Lambda Laufzeit](#) unterstützen, die für die Lambda-Funktionen erforderlich ist, die Sie lokal ausführen möchten. Erforderliche Bibliotheken müssen auf dem Core installiert und zur PATH-Umgebungsvariablen hinzugefügt werden. Auf einem Core können mehrere Bibliotheken installiert werden.

- [Python](#) Version 3.7 für Funktionen, die die Laufzeit Python 3.7 verwenden.
- [Python](#) Version 2.7 für Funktionen, die die Laufzeit Python 2.7 verwenden.
- [Node.js](#)-Version 12.x für Funktionen, die die Node.js 12.x-Laufzeit verwenden.
- [Java](#) Version 8 oder höher für Funktionen, die die Laufzeit Java 8 verwenden.

 Note

Die Ausführung von Java auf einer OpenWrt Distribution wird nicht offiziell unterstützt. Wenn Ihr OpenWrt Build jedoch Java-Unterstützung bietet, können Sie möglicherweise in Java verfasste Lambda-Funktionen auf Ihren Geräten ausführen. OpenWrt

Weitere Informationen zur AWS IoT Greengrass Unterstützung von Lambda-Laufzeiten finden Sie unter. [Lokale Lambda-Funktionen ausführen](#)

- Die folgenden Shell-Befehle (nicht die BusyBox Varianten) werden vom [over-the-air \(OTA-\) Update-Agent](#) benötigt:
 - `wget`
 - `realpath`
 - `tar`
 - `readlink`
 - `basename`
 - `dirname`
 - `pidof`
 - `df`
 - `grep`
 - `umount`
 - `mv`
 - `gzip`
 - `mkdir`
 - `rm`
 - `ln`

- `cut`
- `cat`
- `/bin/bash`

GGC v1.9

Unterstützte Plattformen:

- Architektur: Armv7l
 - Betriebssystem: Linux
 - Betriebssystem: Linux ([OpenWrt](#))
- Architektur: Armv8 () AArch64
 - Betriebssystem: Linux
 - Betriebssystem: Linux ([OpenWrt](#))
- Architektur: Armv6l
 - Betriebssystem: Linux
- Architektur: x86_64
 - Betriebssystem: Linux
- Windows-, MacOS- und Linux-Plattformen können AWS IoT Greengrass in einem Docker-Container ausgeführt werden. Weitere Informationen finden Sie unter [the section called “ AWS IoT Greengrass In einem Docker-Container ausführen”](#).

Voraussetzungen:

- Für die AWS IoT Greengrass Core-Software stehen mindestens 128 MB Festplattenspeicher zur Verfügung. Wenn Sie den [OTA-Update-Agent](#) verwenden, sind mindestens 400 MB erforderlich.
- Der AWS IoT Greengrass Core-Software sind mindestens 128 MB RAM zugewiesen.
- Linux-Kernel-Version:
 - Die Linux-Kernel-Version 4.4 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass mit [Containern](#) zu unterstützen.
 - Die Linux-Kernel-Version 3.17 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass ohne Container zu unterstützen. In dieser Konfiguration muss die standardmäßige Containerisierung der Lambda-Funktion für die Greengrass-Gruppe auf

Kein Container gesetzt werden. Detaillierte Anweisungen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).

- [GNU C Library \(Glibc\)](#) Version 2.14 oder höher. OpenWrt Distributionen benötigen die [Musl C Library](#) Version 1.1.16 oder höher.
- Das Verzeichnis `/var/run` muss auf dem Gerät vorhanden sein.
- Die Dateien `/dev/stdin`, `/dev/stdout` und `/dev/stderr` müssen verfügbar sein.
- Der Hard- und Softlink-Schutz muss auf dem Gerät aktiviert sein. Andernfalls AWS IoT Greengrass kann es nur im unsicheren Modus mit dem Flag ausgeführt werden. `-i`
- Die folgenden Linux-Kernel-Konfigurationen müssen auf dem Gerät aktiviert sein:
 - Namespace
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`
 - `CONFIG_USER_NS`
 - `CONFIG_PID_NS`
 - Cgroups:
 - `CONFIG_CGROUP_DEVICE`
 - `CONFIG_CGROUPS`
 - `CONFIG_MEMCG`

Der Kernel muss [cgroups](#) unterstützen. Bei der Ausführung AWS IoT Greengrass mit [Containern](#) gelten die folgenden Anforderungen:

- Die Speicher-Cgroup muss aktiviert und gemountet sein, AWS IoT Greengrass damit das Speicherlimit für Lambda-Funktionen festgelegt werden kann.
- Die Geräte-Cgroup muss aktiviert und gemountet sein, wenn Lambda-Funktionen mit [lokalem Ressourcenzugriff zum Öffnen von Dateien auf dem AWS IoT Greengrass Kerngerät](#) verwendet werden.
- Weitere:
 - `CONFIG_POSIX_MQUEUE`
 - `CONFIG_OVERLAY_FS`
 - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`
 - `CONFIG_SECCOMP_FILTER`

- CONFIG_SECCOMP
- CONFIG_SHMEM
- Das Stammzertifikat für Amazon S3 AWS IoT muss im System Trust Store vorhanden sein.
- Bibliotheken, die die [AWS Lambda Laufzeit](#) unterstützen, die für die Lambda-Funktionen erforderlich ist, die Sie lokal ausführen möchten. Erforderliche Bibliotheken müssen auf dem Core installiert und zur PATH-Umgebungsvariablen hinzugefügt werden. Auf einem Core können mehrere Bibliotheken installiert werden.
 - [Python](#) Version 2.7 für Funktionen, die die Laufzeit Python 2.7 verwenden.
 - [Python](#) Version 3.7 für Funktionen, die die Laufzeit Python 3.7 verwenden.
 - [Node.js](#) Version 6.10 oder höher für Funktionen, die die Laufzeit Node.js 6.10 verwenden.
 - [Node.js](#) Version 8.10 oder höher für Funktionen, die die Laufzeit Node.js 8.10 verwenden.
 - [Java](#) Version 8 oder höher für Funktionen, die die Laufzeit Java 8 verwenden.

 Note

Die Ausführung von Java auf einer OpenWrt Distribution wird nicht offiziell unterstützt. Wenn Ihr OpenWrt Build jedoch Java-Unterstützung bietet, können Sie möglicherweise in Java verfasste Lambda-Funktionen auf Ihren Geräten ausführen. OpenWrt

Weitere Informationen zur AWS IoT Greengrass Unterstützung von Lambda-Laufzeiten finden Sie unter. [Lokale Lambda-Funktionen ausführen](#)

- Die folgenden Shell-Befehle (nicht die BusyBox Varianten) werden vom [over-the-air \(OTA-\) Update-Agent](#) benötigt:
 - wget
 - realpath
 - tar
 - readlink
 - basename
 - dirname
 - pidof

- `grep`
- `umount`
- `mv`
- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`

GGC v1.8

- Unterstützte Plattformen:
 - Architektur: ARMv7L; Betriebssystem: Linux
 - Architektur: x86_64; Betriebssystem: Linux
 - Architektur: Armv8 (AArch64); Betriebssystem: Linux
 - Windows-, MacOS- und Linux-Plattformen können AWS IoT Greengrass in einem Docker-Container ausgeführt werden. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass In einem Docker-Container ausführen”](#).
 - Linux-Plattformen können AWS IoT Greengrass mit dem Greengrass-Snap, der über [Snapcraft](#) erhältlich ist, eine Version von mit eingeschränkter Funktionalität ausführen. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass Snap-Software”](#).
- Die folgenden Elemente sind erforderlich:
 - Für die AWS IoT Greengrass Core-Software stehen mindestens 128 MB Festplattenspeicher zur Verfügung. Wenn Sie den [OTA-Update-Agent](#) verwenden, sind mindestens 400 MB erforderlich.
 - Der AWS IoT Greengrass Core-Software sind mindestens 128 MB RAM zugewiesen.
 - Linux-Kernel-Version:
 - Die Linux-Kernel-Version 4.4 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass mit [Containern](#) zu unterstützen.

- Die Linux-Kernel-Version 3.17 oder höher ist erforderlich, um die Ausführung AWS IoT Greengrass ohne Container zu unterstützen. In dieser Konfiguration muss die standardmäßige Containerisierung der Lambda-Funktion für die Greengrass-Gruppe auf Kein Container gesetzt werden. Detaillierte Anweisungen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).
- [GNU-C-Bibliothek](#) (glibc) Version 2.14 oder höher.
- Das Verzeichnis `/var/run` muss auf dem Gerät vorhanden sein.
- Die Dateien `/dev/stdin`, `/dev/stdout` und `/dev/stderr` müssen verfügbar sein.
- Der Hard- und Softlink-Schutz muss auf dem Gerät aktiviert sein. Andernfalls AWS IoT Greengrass kann sie nur im unsicheren Modus mit dem Flag ausgeführt werden. `-i`
- Die folgenden Linux-Kernel-Konfigurationen müssen auf dem Gerät aktiviert sein:
 - Namespace
 - `CONFIG_IPC_NS`
 - `CONFIG_UTS_NS`
 - `CONFIG_USER_NS`
 - `CONFIG_PID_NS`
 - Cgroups:
 - `CONFIG_CGROUP_DEVICE`
 - `CONFIG_CGROUPS`
 - `CONFIG_MEMCG`

Der Kernel muss [cgroups](#) unterstützen. Bei der Ausführung AWS IoT Greengrass mit [Containern](#) gelten die folgenden Anforderungen:

- Die Speicher-Cgroup muss aktiviert und gemountet sein, AWS IoT Greengrass damit das Speicherlimit für Lambda-Funktionen festgelegt werden kann.
- Die Geräte-Cgroup muss aktiviert und gemountet sein, wenn Lambda-Funktionen mit [lokalem Ressourcenzugriff zum](#) Öffnen von Dateien auf dem AWS IoT Greengrass Kerngerät verwendet werden.
- Weitere:
 - `CONFIG_POSIX_MQUEUE`
 - `CONFIG_OVERLAY_FS`
 - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`

- CONFIG_SECCOMP_FILTER
- CONFIG_KEYS
- CONFIG_SECCOMP
- CONFIG_SHMEM
- Das Stammzertifikat für Amazon S3 AWS IoT muss im System Trust Store vorhanden sein.
- Die folgenden Elemente sind zwingend erforderlich:
 - Bibliotheken, die die [AWS Lambda Laufzeit](#) unterstützen, die für die Lambda-Funktionen erforderlich ist, die Sie lokal ausführen möchten. Erforderliche Bibliotheken müssen auf dem Core installiert und zur PATH-Umgebungsvariablen hinzugefügt werden. Auf einem Core können mehrere Bibliotheken installiert werden.
 - [Python](#) Version 2.7 für Funktionen, die die Laufzeit Python 2.7 verwenden.
 - [Node.js](#) Version 6.10 oder höher für Funktionen, die die Laufzeit Node.js 6.10 verwenden.
 - [Java](#) Version 8 oder höher für Funktionen, die die Laufzeit Java 8 verwenden.
 - Die folgenden Shell-Befehle (nicht die BusyBox Varianten) werden vom [over-the-air \(OTA-\) Update-Agent](#) benötigt:
 - wget
 - realpath
 - tar
 - readlink
 - basename
 - dirname
 - pidof
 - df
 - grep
 - umount
 - mv
 - gzip
 - mkdir
 - rm
 - ln
 - cut

- `cat`

Informationen zu AWS IoT Greengrass Kontingenten (Limits) finden Sie unter [Service Quotas](#) in der Allgemeine Amazon Web Services-Referenz.

Preisinformationen finden Sie unter [AWS IoT Greengrass -Preise](#) und [AWS IoT Core -Preise](#).

AWS IoT Greengrass lädt herunter

Sie können die folgenden Informationen zum Suchen und Herunterladen von Software für die Verwendung mit AWS IoT Greengrass einsehen.

Themen

- [AWS IoT Greengrass Kernsoftware](#)
- [AWS IoT Greengrass Snap-Software](#)
- [AWS IoT Greengrass Docker-Software](#)
- [AWS IoT Greengrass Kern-SDK](#)
- [Unterstützte Machine Learning-Laufzeiten und -Bibliotheken](#)
- [AWS IoT Greengrass ML SDK-Software](#)

AWS IoT Greengrass Kernsoftware

Die AWS IoT Greengrass Core-Software erweitert die AWS Funktionalität auf ein AWS IoT Greengrass Kerngerät und ermöglicht es lokalen Geräten, lokal auf die von ihnen generierten Daten zu reagieren.

v1.11

1.11.6

Fehlerbehebungen und Verbesserungen:

- Verbesserte Widerstandsfähigkeit bei plötzlichem Stromausfall während eines Einsatzes.
- Es wurde ein Problem behoben, bei dem die Beschädigung von Stream Manager-Daten dazu führen konnte, dass die AWS IoT Greengrass Core-Software nicht gestartet werden konnte.

- Es wurde ein Problem behoben, bei dem neue Client-Geräte in bestimmten Szenarien keine Verbindung zum Core herstellen konnten.
- Es wurde ein Problem behoben, bei dem Stream-Manager-Streamnamen nicht enthalten konnten. `.log`.

1.11.5

Fehlerbehebungen und Verbesserungen:

- Allgemeine Leistungssteigerungen und Bugfixes.

1.11.4

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem mit dem Stream Manager behoben, das Upgrades auf die AWS IoT Greengrass Core-Software v1.11.3 verhinderte. Wenn Sie den Stream Manager verwenden, um Daten in die Cloud zu exportieren, können Sie jetzt ein OTA-Update verwenden, um eine frühere v1.x-Version der AWS IoT Greengrass Core-Software auf v1.11.4 zu aktualisieren.
- Allgemeine Leistungssteigerungen und Bugfixes.

1.11.3

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem behoben, das dazu führte, dass die AWS IoT Greengrass Core-Software, die auf einem Ubuntu-Gerät im Handumdrehen ausgeführt wurde, nach einem plötzlichen Stromausfall des Geräts nicht mehr reagierte.
- Es wurde ein Problem behoben, das zu einer verzögerten Übermittlung von MQTT-Nachrichten an langlebige Lambda-Funktionen führte.
- Es wurde ein Problem behoben, das dazu führte, dass MQTT-Nachrichten nicht korrekt gesendet wurden, wenn der `maxWorkItemCount` Wert auf einen Wert größer als gesetzt wurde. 1024
- Es wurde ein Problem behoben, das dazu führte, dass der OTA-Update-Agent den in der `keepAlive` Eigenschaft in angegebenen KeepAlive MQTT-Zeitraum ignorierte. [config.json](#)
- Allgemeine Leistungssteigerungen und Bugfixes.

Important

Wenn Sie Stream Manager verwenden, um Daten in die Cloud zu exportieren, führen Sie kein Upgrade von einer früheren Version AWS IoT Greengrass v1.x auf die Core-

Software v1.11.3 durch. Wenn Sie den Stream Manager zum ersten Mal aktivieren, empfehlen wir dringend, zuerst die neueste Version der AWS IoT Greengrass Core-Software zu installieren.

1.11.1

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem behoben, das zu einem erhöhten Speicherverbrauch für den Stream-Manager führte.
- Es wurde ein Problem behoben, das dazu führte, dass der Stream-Manager die Sequenznummer des Streams auf zurücksetzte, 0 wenn das Greengrass-Core-Gerät länger als den angegebenen Zeitraum time-to-live (TTL) der Stream-Daten ausgeschaltet war.
- Es wurde ein Problem behoben, das verhinderte, dass der Stream Manager Wiederholungsversuche, Daten in den zu exportieren, korrekt unterbrach. AWS Cloud

1.11.0

Neue Funktionen:

- Ein Telemetrieagent auf dem Greengrass-Core sammelt lokale Telemetriedaten und veröffentlicht sie an. AWS Cloud Um die Telemetriedaten für die weitere Verarbeitung abzurufen, können Kunden eine EventBridge Amazon-Regel erstellen und ein Ziel abonnieren. Weitere Informationen finden Sie unter [Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten](#).
- Eine lokale HTTP-API gibt eine Momentaufnahme des aktuellen Status der lokalen Arbeitsprozesse zurück, die von gestartet wurden AWS IoT Greengrass. Weitere Informationen finden Sie unter [Aufrufen der lokalen Health Check-API](#).
- Ein [Stream-Manager](#) exportiert automatisch Daten nach Amazon S3 und AWS IoT SiteWise.

Mit neuen [Stream-Manager-Parametern](#) können Sie bestehende Streams aktualisieren und den Datenexport anhalten oder fortsetzen.

- Support für die Ausführung von Python 3.8.x Lambda-Funktionen auf dem Kern.
- Eine neue ggDaemonPort Eigenschaft [config.json](#), die verwendet wird, um die Greengrass-Core-IPC-Portnummer zu konfigurieren. Die Standard-Portnummer ist 8000.

Eine neue `systemComponentAuthTimeout` Eigenschaft, mit der Sie [config.json](#) das Timeout für die Greengrass-Core-IPC-Authentifizierung konfigurieren. Das Standard-Timeout beträgt 5000 Millisekunden.

- Die maximale Anzahl von AWS IoT Geräten pro AWS IoT Greengrass Gruppe wurde von 200 auf 2500 erhöht.

Die maximale Anzahl von Abonnements pro Gruppe wurde von 1000 auf 10000 erhöht.

Weitere Informationen finden Sie unter [AWS IoT Greengrass -Endpunkte und -Kontingente](#).

Fehlerbehebungen und Verbesserungen:

- Allgemeine Optimierung, die die Speicherauslastung der Greengrass-Serviceprozesse reduzieren kann.
- Ein neuer Laufzeitkonfigurationsparameter (`mountAllBlockDevices`) ermöglicht es Greengrass, Bind-Mounts zu verwenden, um alle Blockgeräte nach der Einrichtung von OverlayFS in einen Container zu mounten. Mit dieser Funktion wurde ein Problem behoben, das dazu führte, dass die Greengrass-Bereitstellung fehlschlug, wenn es sich `/usr` nicht in der `/` Hierarchie befindet.
- Es wurde ein Problem behoben, das zu einem AWS IoT Greengrass Kernausfall führte, wenn `/tmp` es sich um einen Symlink handelt.
- Es wurde ein Problem behoben, durch das der Greengrass Deployment Agent ungenutzte Modellartefakte für maschinelles Lernen aus dem `mlmodel_public` Ordner entfernen konnte.
- Allgemeine Leistungssteigerungen und Bugfixes.

Um die AWS IoT Greengrass Core-Software auf Ihrem Core-Gerät zu installieren, laden Sie das Paket für Ihre Architektur und Ihr Betriebssystem (OS) herunter und folgen Sie dann den Schritten im [Handbuch Erste Schritte](#).

 Tip

AWS IoT Greengrass bietet auch andere Optionen für die Installation der AWS IoT Greengrass Core-Software. Sie können beispielsweise das [Greengrass-Geräte-Setup](#) verwenden, um Ihre Umgebung zu konfigurieren und die neueste Version der AWS IoT Greengrass Core-Software zu installieren. Oder Sie können auf unterstützten Debian-Plattformen den [APT-Paketmanager](#) verwenden, um die AWS IoT Greengrass Core-

Software zu installieren oder zu aktualisieren. Weitere Informationen finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

Architektur	Betriebssystem	Link
Armv8 () AArch64	Linux	Download
Arm 8 () AArch64	Linux () OpenWrt	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Download
x86_64	Linux	Download

Extended life versions

1.10.5

Neue Funktionen in Version 1.10:

- Ein Stream-Manager, der Datenströme lokal verarbeitet und sie AWS Cloud automatisch in den exportiert. Diese Funktion erfordert Java 8 auf dem Greengrass-Core-Gerät. Weitere Informationen finden Sie unter [Verwalten von Daten-Streams](#).
- Ein neuer Greengrass-Docker-Anwendungsbereitstellungs-Konnektor, der eine Docker-Anwendung auf einem Core-Gerät ausführt. Weitere Informationen finden Sie unter [the section called “Docker-Anwendungsbereitstellung”](#).
- Ein neuer SiteWise IoT-Connector, der industrielle Gerätedaten von OPC-UA-Servern an Anlagenimmobilien in sendet. AWS IoT SiteWise Weitere Informationen finden Sie unter [the section called “IoT SiteWise”](#).
- Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, können auf Ressourcen für maschinelles Lernen in der Greengrass-Gruppe zugreifen. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Machine Learning-Ressourcen”](#).

- Support für persistente MQTT-Sitzungen mit AWS IoT. Weitere Informationen finden Sie unter [the section called “Persistente MQTT-Sitzungen mit AWS IoT Core”](#).
- Lokaler MQTT-Datenverkehr kann über einen anderen Port als den Standard-Port 8883 übertragen. Weitere Informationen finden Sie unter [the section called “MQTT-Port für lokales Messaging”](#).
- Neue queueFullPolicy Optionen im [AWS IoT Greengrass Core SDK](#) für die zuverlässige Veröffentlichung von Nachrichten über Lambda-Funktionen.
- Support für die Ausführung von Node.js 12.x Lambda-Funktionen auf dem Kern.

Fehlerbehebungen und Verbesserungen:

- Over-the-air (OTA) -Updates mit Hardware-Sicherheitsintegration können mit OpenSSL 1.1 konfiguriert werden.
- [Stream-Manager](#) ist widerstandsfähiger gegen Dateidatenbeschädigung.
- Es wurde ein Problem behoben, durch das auf Geräten mit Linux-Kernel 5.1 und höher ein Sysfs-Mount-Fehler verursacht wurde.
- Eine neue mqttOperationTimeout Eigenschaft in [config.json](#), mit der Sie das Timeout für Veröffentlichungs-, Abonnier- und Abmeldevorgänge in MQTT-Verbindungen festlegen. AWS IoT Core
- Es wurde ein Problem behoben, das zu einem erhöhten Speicherverbrauch für den Stream-Manager führte.
- Eine neue systemComponentAuthTimeout Eigenschaft, mit der Sie [config.json](#) das Timeout für die Greengrass-Core-IPC-Authentifizierung konfigurieren. Das Standard-Timeout beträgt 5000 Millisekunden.
- Es wurde ein Problem behoben, das dazu führte, dass der OTA-Update-Agent den in der Eigenschaft in angegebenen KeepAlive MQTT-Zeitraum ignorierte. keepAlive [config.json](#)
- Es wurde ein Problem behoben, das dazu führte, dass MQTT-Nachrichten nicht korrekt gesendet wurden, wenn der maxWorkItemCount Wert auf einen Wert größer als gesetzt wurde. 1024
- Es wurde ein Problem behoben, das zu einer verzögerten Übermittlung von MQTT-Nachrichten an langlebige Lambda-Funktionen führte.
- Es wurde ein Problem behoben, das dazu führte, dass die AWS IoT Greengrass Core-Software, die auf einem Ubuntu-Gerät im Handumdrehen ausgeführt wurde, nach einem plötzlichen Stromausfall des Geräts nicht mehr reagierte.

- Allgemeine Leistungssteigerungen und Bugfixes.

Um die AWS IoT Greengrass Core-Software auf Ihrem Core-Gerät zu installieren, laden Sie das Paket für Ihre Architektur und Ihr Betriebssystem (OS) herunter und folgen Sie dann den Schritten im [Handbuch Erste Schritte](#).

Architektur	Betriebssystem	Link
Armv8 () AArch64	Linux	Download
Arm 8 () AArch64	Linux () OpenWrt	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Download
x86_64	Linux	Download

1.9.4

Neue Funktionen in Version 1.9:

- Support für Lambda-Laufzeiten von Python 3.7 und Node.js 8.10. Lambda-Funktionen, die die Laufzeiten Python 3.7 und Node.js 8.10 verwenden, können jetzt auf einem Core ausgeführt werden. AWS IoT Greengrass (unterstützt AWS IoT Greengrass weiterhin die Laufzeiten Python 2.7 und Node.js 6.10.)
- Optimierte MQTT-Verbindungen. Der Greengrass Core baut weniger Verbindungen mit dem AWS IoT Core auf. Diese Änderung kann Betriebskosten hinsichtlich Gebühren, die auf der Anzahl der Verbindungen basieren, senken.
- Elliptic Curve (EC)-Schlüssel für den lokalen MQTT-Server. Der lokale MQTT-Server unterstützt EC-Schlüssel zusätzlich zu den RSA-Schlüsseln. (Das MQTT-Serverzertifikat verfügt über eine SHA-256-RSA-Signatur, unabhängig vom Schlüsseltyp.) Weitere Informationen finden Sie unter [the section called “Sicherheitsprinzipale”](#).
- Support für [OpenWrt](#). AWS IoT Greengrass Die Kernsoftware v1.9.2 oder höher kann auf OpenWrt Distributionen mit Armv8 (AArch64) - und ARMv7L-Architekturen installiert werden. Unterstützt derzeit keine ML-Inferenz. OpenWrt

- Support für ARMv6L. AWS IoT Greengrass Die Kernsoftware v1.9.3 oder höher kann auf Raspbian-Distributionen auf ARMv6L-Architekturen (z. B. auf Raspberry Pi Zero-Geräten) installiert werden.
- OTA-Updates auf Port 443 mit ALPN. Greengrass-Kerne, die Port 443 für MQTT-Verkehr verwenden, unterstützen jetzt over-the-air (OTA) -Softwareupdates. AWS IoT Greengrass verwendet die TLS-Erweiterung Application Layer Protocol Network (ALPN), um diese Verbindungen zu aktivieren. Weitere Informationen erhalten Sie unter [OTA-Updates der AWS IoT Greengrass Core-Software](#) und [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).

Um die AWS IoT Greengrass Core-Software auf Ihrem Core-Gerät zu installieren, laden Sie das Paket für Ihre Architektur und Ihr Betriebssystem (OS) herunter und folgen Sie dann den Schritten im [Handbuch Erste Schritte](#).

Architektur	Betriebssystem	Link
Armv8 () AArch64	Linux	Download
Arm 8 () AArch64	Linux () OpenWrt	Download
Armv7l	Linux	Download
Armv7l	Linux (OpenWrt)	Download
Armv6l	Linux	Download
x86_64	Linux	Download

1.8.4

- Neue Funktionen:
 - Konfigurierbare Standardzugriffsidentität für Lambda-Funktionen in der Gruppe. Diese Einstellung auf Gruppenebene bestimmt die Standardberechtigungen, die zum Ausführen von Lambda-Funktionen verwendet werden. Sie können die Benutzer-ID, Gruppen-ID oder beides festlegen. Einzelne Lambda-Funktionen können die Standardzugriffsidentität ihrer Gruppe überschreiben. Weitere Informationen finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

- HTTPS-Datenverkehr über Port 443. HTTPS-Kommunikation kann so konfiguriert werden, dass sie über Port 443 anstatt über Standard-Port 8443 ausgeführt wird. Dies ergänzt die AWS IoT Greengrass Unterstützung für die TLS-Erweiterung Application Layer Protocol Network (ALPN) und ermöglicht es dem gesamten Greengrass-Messaging-Verkehr — sowohl MQTT als auch HTTPS —, Port 443 zu verwenden. Weitere Informationen finden Sie unter [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).
- IDs AWS IoT Vorhersehbar benannter Client für Verbindungen. Diese Änderung ermöglicht die Unterstützung für AWS IoT Device Defender und [AWS IoT -Lebenszyklus-Ereignisse](#), damit Sie Benachrichtigungen für Verbindungs-, Abmelde-, Abonnement- und Abonnementbeendungsereignisse erhalten. Eine vorhersehbare Benennung erleichtert auch die Erstellung von Logik rund um die Verbindung IDs (z. B. die Erstellung von [Abonnementrichtlinienvorlagen](#) auf der Grundlage von Zertifikatsattributen). Weitere Informationen finden Sie unter [the section called “Client IDs für MQTT-Verbindungen mit AWS IoT”](#).

Fehlerbehebungen und Verbesserungen:

- Es wurde ein Problem mit der Schattensynchronisierung und der erneuten Verbindung des Gerätezertifikatmanagers behoben.
- Allgemeine Leistungssteigerungen und Bugfixes.

Um die AWS IoT Greengrass Core-Software auf Ihrem Hauptgerät zu installieren, laden Sie das Paket für Ihre Architektur und Ihr Betriebssystem (OS) herunter und folgen Sie dann den Schritten im [Handbuch Erste Schritte](#).

Architektur	Betriebssystem	Link
Armv8 () AArch64	Linux	Download
Armv7l	Linux	Download
x86_64	Linux	Download

Mit dem Download dieser Software stimmen Sie der [Greengrass Core-Software-Lizenzvereinbarung](#) zu.

Informationen zu anderen Optionen für die Installation der AWS IoT Greengrass Core-Software auf Ihrem Gerät finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

AWS IoT Greengrass Snap-Software

AWS IoT Greengrass Snap 1.11.x ermöglicht es Ihnen, eine eingeschränkte Version von AWS IoT Greengrass über praktische Softwarepakete zusammen mit allen erforderlichen Abhängigkeiten in einer containerisierten Umgebung auszuführen.

Note

Der AWS IoT Greengrass Snap ist für AWS IoT Greengrass die Core-Software v1.11.x verfügbar. AWS IoT Greengrass bietet keinen Snap für v1.10.x. Nicht unterstützte Versionen erhalten keine Bugfixes oder Updates.

Der AWS IoT Greengrass Snap unterstützt keine Konnektoren und maschinelles Lernen (ML) -Inferenz.

Weitere Informationen finden Sie unter [the section called “Lauf AWS IoT Greengrass im Handumdrehen”](#).

AWS IoT Greengrass Docker-Software

AWS stellt ein Dockerfile und Docker-Images bereit, die Ihnen die Ausführung AWS IoT Greengrass in einem Docker-Container erleichtern.

Dockerfile

Dockerfiles enthalten Quellcode zum Erstellen benutzerdefinierter Container-Images. AWS IoT Greengrass Images können so geändert werden, dass es auf unterschiedlichen Plattformarchitekturen ausgeführt werden kann. Es kann auch verkleinert werden. Weitere Anweisungen finden Sie in der README-Datei.

Laden Sie Ihre Zielversion der AWS IoT Greengrass Core-Software herunter.

v1.11

- [Dockerfile für AWS IoT Greengrass v1.11.6](#).

Extended life versions

v1.10

[Dockerfile](#) für v1.10.5. AWS IoT Greengrass

v1.9

[Dockerfile](#) für v1.9.4. AWS IoT Greengrass

v1.8

[Dockerfile](#) für v1.8.1. AWS IoT Greengrass

Docker-Image

Bei Docker-Images sind die AWS IoT Greengrass Core-Software und die zugehörigen Abhängigkeiten auf den Basis-Images von Amazon Linux 2 (x86_64) und Alpine Linux (x86_64, ARMv7L oder) installiert. AArch64 Sie können vorgefertigte Bilder verwenden, um mit AWS IoT Greengrass zu experimentieren.

Important

Am 30. Juni 2022 AWS IoT Greengrass wurde die Wartung für Docker-Images der AWS IoT Greengrass Core-Software v1.x eingestellt, die in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub veröffentlicht wurden. Sie können diese Docker-Images bis zum 30. Juni 2023, also 1 Jahr nach Ende der Wartung, weiterhin von Amazon ECR und Docker Hub herunterladen. Nach dem Ende der Wartung am 30. Juni 2022 erhalten die Docker-Images der AWS IoT Greengrass Core-Software v1.x jedoch keine Sicherheitspatches oder Bugfixes mehr. Wenn Sie einen Produktions-Workload ausführen, der von diesen Docker-Images abhängt, empfehlen wir Ihnen, Ihre eigenen Docker-Images mithilfe der bereitgestellten Dockerfiles zu erstellen. AWS IoT Greengrass Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Laden Sie ein vorgefertigtes Image von [Docker Hub](#) oder Amazon Elastic Container Registry (Amazon ECR) herunter.

- Verwenden Sie für Docker Hub das **version** Tag, um eine bestimmte Version des Greengrass Docker-Images herunterzuladen. Um Tags für alle verfügbaren Images zu finden, überprüfen Sie die Seite Tags im Docker Hub.
- Verwenden Sie für Amazon ECR das **latest** Tag, um die neueste verfügbare Version des Greengrass Docker-Images herunterzuladen. Weitere Informationen zum Auflisten verfügbarer Bildversionen und zum Herunterladen von Bildern von Amazon ECR finden Sie unter [Wird AWS IoT Greengrass in einem Docker-Container ausgeführt.](#)

 Warning

Ab Version 1.11.6 der AWS IoT Greengrass Core-Software enthalten die Greengrass Docker-Images Python 2.7 nicht mehr, da Python 2.7 end-of-life im Jahr 2020 erreicht wurde und keine Sicherheitsupdates mehr erhält. Wenn Sie sich für ein Update auf diese Docker-Images entscheiden, empfehlen wir Ihnen, zu überprüfen, ob Ihre Anwendungen mit den neuen Docker-Images funktionieren, bevor Sie die Updates auf Produktionsgeräten bereitstellen. Wenn Sie Python 2.7 für Ihre Anwendung benötigen, die ein Greengrass Docker-Image verwendet, können Sie das Greengrass Dockerfile so ändern, dass es Python 2.7 für Ihre Anwendung enthält.

AWS IoT Greengrass stellt keine Docker-Images für die Core-Software v1.11.1 bereit. AWS IoT Greengrass

 Note

Standardmäßig können `alpine-aarch64`- und `alpine-armv7l`-Images nur auf ARM-basierten Hosts ausgeführt werden. Um diese Images auf einem x86-Host auszuführen, können Sie [QEMU](#) installieren und die QEMU-Bibliotheken auf dem Host mounten.

Beispiel:

```
docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
```

AWS IoT Greengrass Kern-SDK

Lambda-Funktionen verwenden das AWS IoT Greengrass Core SDK, um lokal mit dem AWS IoT Greengrass Core zu interagieren. Auf diese Weise können bereitgestellte Lambda-Funktionen:

- Tauschen Sie MQTT-Nachrichten aus mit. AWS IoT Core
- Tauschen Sie MQTT-Nachrichten mit Konnektoren, Client-Geräten und anderen Lambda-Funktionen in der Greengrass-Gruppe aus.
- Interagieren mit dem lokalen Shadow-Service.
- Rufen Sie andere lokale Lambda-Funktionen auf.
- Zugreifen auf [geheime Ressourcen](#).
- Interagieren Sie mit dem [Stream-Manager](#).

Laden Sie das AWS IoT Greengrass Core SDK für Ihre Sprache oder Plattform von [herunter](#). GitHub

- [AWS IoT Greengrass Kern-SDK SDK for Java](#)
- [AWS IoT Greengrass Kern-SDK für Node.js](#)
- [AWS IoT Greengrass Kern-SDK für Python](#)
- [AWS IoT Greengrass Kern-SDK für C](#)

Weitere Informationen finden Sie unter [AWS IoT Greengrass Kern-SDK](#).

Unterstützte Machine Learning-Laufzeiten und -Bibliotheken

Um auf einem Greengrass Core [Inferenz durchzuführen](#), müssen Sie die Machine Learning-Laufzeit oder -Bibliothek für Ihren ML-Modelltyp installieren.

AWS IoT Greengrass unterstützt die folgenden ML-Modelltypen. Verwenden Sie diese Links, um Informationen zur Installation der Laufzeit oder Bibliothek für Ihren Modelltyp und Ihre Geräteplattform zu erhalten.

- [Deep Learning Runtime \(DLR\)](#)
- [MXNet](#)
- [TensorFlow](#)

Beispiele für Machine Learning

AWS IoT Greengrass stellt Beispiele bereit, die Sie mit unterstützten ML-Laufzeiten und -Bibliotheken verwenden können. Diese Beispiele werden im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) bereitgestellt.

Deep learning runtime (DLR)

Laden Sie das Beispiel für Ihre Geräteplattform herunter:

- DLR-Beispiel für [Raspberry Pi](#)
- [DLR-Beispiel für NVIDIA Jetson TX2](#)
- DLR-Beispiel für [Intel Atom](#)

Ein Tutorial, das das DLR-Beispiel verwendet, finden Sie unter [the section called “So konfigurieren Sie optimierte Machine Learning-Inferenz”](#).

MXNet

Laden Sie das Beispiel für Ihre Geräteplattform herunter:

- MXNet [Beispiel für Raspberry Pi](#)
- MXNet Beispiel für [NVIDIA Jetson TX2](#)
- MXNet Beispiel für [Intel Atom](#)

Ein Tutorial, das das MXNet Beispiel verwendet, finden Sie unter [the section called “So konfigurieren Sie Machine Learning-Inferenz”](#).

TensorFlow

Laden Sie das [Tensorflow-Beispiel](#) für Ihre Geräteplattform herunter. Dieses Beispiel funktioniert mit Raspberry Pi, NVIDIA Jetson TX2 und Intel Atom.

AWS IoT Greengrass ML SDK-Software

[AWS IoT Greengrass SDK für Machine Learning](#) Dadurch können die von Ihnen Lambda Funktionen ein lokales Machine-Learning-Modell verwenden und Daten zum Hochladen und Veröffentlichen an den [ML-Feedback-Connector](#) senden.

v1.1.0

- [Python 3.7.](#)

v1.0.0

- [Python 2.7.](#)

Bitte geben Sie uns Feedback

Wir freuen uns über Ihr Feedback. [Um uns zu kontaktieren, besuche AWS re:POST und verwende das AWS IoT Greengrass Tag.](#)

Installieren Sie die AWS IoT Greengrass Core-Software

Die AWS IoT Greengrass Core-Software erweitert die AWS Funktionalität auf ein AWS IoT Greengrass Kerngerät und ermöglicht es lokalen Geräten, lokal auf die von ihnen generierten Daten zu reagieren.

AWS IoT Greengrass bietet mehrere Optionen für die Installation der AWS IoT Greengrass Core-Software:

- [Laden Sie eine tar.gz-Datei herunter und extrahieren Sie sie.](#)
- [Führen Sie das Greengrass Device Setup-Skript](#) aus.
- [Installieren Sie aus einem APT-Repository.](#)

AWS IoT Greengrass bietet auch containerisierte Umgebungen, in denen die AWS IoT Greengrass Core-Software ausgeführt wird.

- [AWS IoT Greengrass In einem Docker-Container ausführen.](#)
- [AWS IoT Greengrass Im Handumdrehen ausführen.](#)

Laden Sie das AWS IoT Greengrass Core-Softwarepaket herunter und extrahieren Sie es

Wählen Sie die AWS IoT Greengrass Core-Software für Ihre Plattform aus, um sie als Datei tar.gz herunterzuladen und auf Ihr Gerät zu extrahieren. Sie können aktuelle Versionen der Software herunterladen. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass Kernsoftware”](#).

Ausführen des Greengrass Device Setup-Skripts

Führen Sie das Greengrass-Geräte-Setup aus, um Ihr Gerät zu konfigurieren, die neueste AWS IoT Greengrass Core-Softwareversion zu installieren und innerhalb weniger Minuten eine Hello World Lambda-Funktion bereitzustellen. Weitere Informationen finden Sie unter [the section called “Schnellstart: Greengrass-Geräteeinrichtung”](#).

Installieren Sie die AWS IoT Greengrass Core-Software aus einem APT-Repository

Important

Ab dem 11. Februar 2022 können Sie die AWS IoT Greengrass Core-Software nicht mehr über ein APT-Repository installieren oder aktualisieren. Auf Geräten, auf denen Sie das AWS IoT Greengrass Repository hinzugefügt haben, müssen Sie [das Repository aus der Quellenliste entfernen](#). Geräte, auf denen die Software aus dem APT-Repository ausgeführt wird, funktionieren weiterhin normal. Wir empfehlen, dass Sie die AWS IoT Greengrass Core-Software mithilfe von [Tar-Dateien](#) aktualisieren.

Das von bereitgestellte APT-Repository AWS IoT Greengrass umfasst die folgenden Pakete:

- `aws-iot-greengrass-core`. Installiert die AWS IoT Greengrass Core-Software.
- `aws-iot-greengrass-keyring`. Installiert die GnuPG-Schlüssel (GPG), die zum Signieren des AWS IoT Greengrass Paket-Repositorys verwendet werden.

Mit dem Download dieser Software stimmen Sie der [Greengrass Core-Software-Lizenzvereinbarung](#) zu.

Topics

- [Verwenden von systemd-Skripts zum Verwalten des Greengrass Daemon-Lebenszyklus](#)
- [Deinstallieren Sie die AWS IoT Greengrass Kernsoftware mithilfe des APT-Repositorys](#)
- [Entfernen Sie die Quellen des AWS IoT Greengrass Kern-Software-Repositorys](#)

Verwenden von systemd-Skripts zum Verwalten des Greengrass Daemon-Lebenszyklus

Das `aws-iot-greengrass-core` Paket installiert auch systemd Skripten, mit denen Sie den Lebenszyklus der AWS IoT Greengrass Core-Software (Daemon) verwalten können.

- So starten Sie den Greengrass-Daemon während des Boots:

```
systemctl enable greengrass.service
```

- So starten Sie den Greengrass-Daemon:

```
systemctl start greengrass.service
```

- So halten Sie den Greengrass-Daemon an:

```
systemctl stop greengrass.service
```

- So überprüfen Sie den Status des Greengrass-Daemons:

```
systemctl status greengrass.service
```

Deinstallieren Sie die AWS IoT Greengrass Kernsoftware mithilfe des APT-Repositorys

Wenn Sie die AWS IoT Greengrass Kernsoftware deinstallieren, können Sie wählen, ob Sie die Konfigurationsinformationen der AWS IoT Greengrass Kernsoftware wie Gerätezertifikate, Gruppeninformationen und Protokolldateien beibehalten oder entfernen möchten.

Um die AWS IoT Greengrass Kernsoftware zu deinstallieren und die Konfigurationsinformationen beizubehalten

- Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Kernsoftwarepakete zu entfernen und die Konfigurationsinformationen im `/greengrass` Ordner beizubehalten.

```
sudo apt remove aws-iot-greengrass-core aws-iot-greengrass-keyring
```

Um die AWS IoT Greengrass Kernsoftware zu deinstallieren und die Konfigurationsinformationen zu entfernen

1. Führen Sie den folgenden Befehl aus, um die AWS IoT Greengrass Kernsoftwarepakete und die Konfigurationsinformationen aus dem `zu entfernen/greengrass` folder.

```
sudo apt purge aws-iot-greengrass-core aws-iot-greengrass-keyring
```

2. Entfernen Sie das AWS IoT Greengrass Core-Software-Repository aus Ihrer Quellenliste. Weitere Informationen finden Sie unter [Entfernen Sie die Quellen des AWS IoT Greengrass Kern-Software-Repositorys](#).

Entfernen Sie die Quellen des AWS IoT Greengrass Kern-Software-Repositorys

Sie können die AWS IoT Greengrass Kern-Software-Repository-Quellen entfernen, wenn Sie die AWS IoT Greengrass Kernsoftware nicht mehr aus dem APT-Repository installieren oder aktualisieren müssen. Nach dem 11. Februar 2022 müssen Sie das Repository aus Ihrer Quellenliste entfernen, um einen Fehler bei der Ausführung zu vermeiden `apt update`.

Um das APT-Repository aus der Quellenliste zu entfernen

- Führen Sie die folgenden Befehle aus, um das AWS IoT Greengrass Core-Software-Repository aus der Quellenliste zu entfernen.

```
sudo rm /etc/apt/sources.list.d/greengrass.list  
sudo apt update
```

AWS IoT Greengrass In einem Docker-Container ausführen

AWS IoT Greengrass stellt ein Dockerfile und Docker-Images bereit, die es Ihnen erleichtern, die AWS IoT Greengrass Core-Software in einem Docker-Container auszuführen. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass Docker-Software”](#).

Note

Sie können eine Docker-Anwendung auch auf einem Greengrass-Kerngerät ausführen. Verwenden Sie dazu den [Greengrass-Docker-Anwendungsbereitstellungskonnektor](#).

Im Handumdrehen ausführen AWS IoT Greengrass

AWS IoT Greengrass Snap 1.11.x ermöglicht es Ihnen, eine eingeschränkte Version von AWS IoT Greengrass über praktische Softwarepakete zusammen mit allen erforderlichen Abhängigkeiten in einer containerisierten Umgebung auszuführen.

[Am 31. Dezember 2023 AWS IoT Greengrass wird die Wartung für die AWS IoT Greengrass Kernsoftwareversion 1.11.x Snap, die auf snapcraft.io veröffentlicht ist, eingestellt.](#) Geräte, auf denen der Snap derzeit ausgeführt wird, funktionieren bis auf weiteres weiter. Nach Ende der Wartung erhält der AWS IoT Greengrass Core-Snap jedoch keine Sicherheitspatches oder Bugfixes mehr.

Snap-Konzepte

Im Folgenden finden Sie wichtige Snap-Konzepte, die Ihnen helfen sollen, die Verwendung von AWS IoT Greengrass Snap zu verstehen:

Channel

Eine Snap-Komponente, die definiert, welche Version eines Snaps installiert ist, und deren Aktualisierung überwacht wird. Snaps werden automatisch auf die neueste Version des aktuellen Kanals aktualisiert.

Schnittstelle

Eine Snap-Komponente, die Zugriff auf Ressourcen wie Netzwerke und Benutzerdateien gewährt.

Um den AWS IoT Greengrass Snap auszuführen, müssen die folgenden Schnittstellen angeschlossen sein. Beachten Sie, dass die Verbindung zuerst hergestellt und niemals getrennt werden `greengrass-support-no-container` muss.

- **greengrass-support-no-container**
- hardware-observe
- home-for-hooks
- hugepages-control
- log-observe
- mount-observe
- network
- network-bind
- network-control
- process-control
- system-observe

Die anderen Schnittstellen sind optional. Wenn Ihre Lambda-Funktionen Zugriff auf bestimmte Ressourcen benötigen, müssen Sie möglicherweise eine Verbindung zu den entsprechenden Schnittstellen herstellen.

Aktualisieren

Snapd werden automatisch aktualisiert. Der snapd Daemon ist der Snap-Paketmanager, der standardmäßig viermal täglich nach Updates sucht. Jede Aktualisierungsprüfung wird als Aktualisierung bezeichnet. Bei einer Aktualisierung stoppt der Daemon, der Snap wird aktualisiert und anschließend wird der Daemon neu gestartet.

[Weitere Informationen finden Sie auf der Snapcraft-Website.](#)

Was ist neu in AWS IoT Greengrass Snap v1.11.x

Im Folgenden wird beschrieben, was mit der Version 1.11.x des Snap neu und geändert wurde. AWS IoT Greengrass

- Diese Version unterstützt nur den `snap_daemon` Benutzer, der als Benutzer-ID (UID) und Gruppe (GID) angezeigt wird. 584788
- Diese Version unterstützt nur nicht containerisierte Lambda-Funktionen.

 Important

Da nicht containerisierte Lambda-Funktionen denselben Benutzer (snap_daemon) gemeinsam nutzen müssen, sind die Lambda-Funktionen nicht voneinander isoliert. Weitere Informationen finden Sie unter [Steuern der Ausführung von Greengrass Lambda-Funktionen mithilfe einer gruppenspezifischen](#) Konfiguration.

- Diese Version unterstützt die Laufzeiten C, C++, Java 8, Node.js 12.x, Python 2.7, Python 3.7 und Python 3.8.

 Note

Um redundante Python-Laufzeiten zu vermeiden, führen Lambda-Funktionen von Python 3.7 tatsächlich die Python 3.8-Laufzeit aus.

Erste Schritte mit Snap AWS IoT Greengrass

Das folgende Verfahren hilft Ihnen bei der Installation und Konfiguration des AWS IoT Greengrass Snaps auf Ihrem Gerät.

Voraussetzungen

Um den AWS IoT Greengrass Snap auszuführen, müssen Sie wie folgt vorgehen:

- Führen Sie den AWS IoT Greengrass Snap auf einer unterstützten Linux-Distribution wie Ubuntu, Linux Mint, Debian und Fedora aus.
- Installieren Sie den snapd Daemon auf Ihrem Gerät. Der snapd Daemon einschließlich des snap Tools verwaltet die Snap-Umgebung auf Ihrem Gerät.

Eine Liste der unterstützten Linux-Distributionen und Installationsanweisungen finden Sie in der Snap-Dokumentation unter [Snapd installieren](#).

Installieren und konfigurieren Sie den Snap AWS IoT Greengrass

Das folgende Tutorial zeigt Ihnen, wie Sie den AWS IoT Greengrass Snap auf Ihrem Gerät installieren und konfigurieren.

 Note

- Obwohl dieses Tutorial eine Amazon EC2 EC2-Instance (x86 t2.micro Ubuntu 20.04) verwendet, können Sie den AWS IoT Greengrass Snap mit physischer Hardware wie einem Raspberry Pi ausführen.
- Der snapd Daemon ist auf Ubuntu vorinstalliert.

1. Installieren Sie den core18 Snap, indem Sie den folgenden Befehl im Terminal Ihres Geräts ausführen:

```
sudo snap install core18
```

Der core18 Snap ist ein [Basis-Snap](#), der eine Laufzeitumgebung mit häufig verwendeten Bibliotheken bietet. Dieser Snap wurde mit [Ubuntu 18.04](#) LTS erstellt.

2. Führen Sie das Upgrade snapd durch, indem Sie den folgenden Befehl ausführen:

```
sudo snap install --channel=edge snapd; sudo snap refresh --channel=edge snapd
```

3. Führen Sie den `snap list` Befehl aus, um zu überprüfen, ob Sie den AWS IoT Greengrass Snap installiert haben.

Die folgende Beispielantwort zeigt, dass dieser installiert snapd ist, aber `aws-iot-greengrass` nicht.

Name	Version	Rev	Tracking	Publisher	Notes
amazon-ssm-agent	3.0.161.0	2996	latest/stable/...	aws#	classic
core	16-2.48	10444	latest/stable	canonical#	core
core18	20200929	1932	latest/stable	canonical#	base
lxd	4.0.4	18150	4.0/stable/...	canonical#	-
snapd	2.48+git548.g929ccfb	10526	latest/edge	canonical#	snapd

4. Wählen Sie eine der folgenden Optionen, um AWS IoT Greengrass Snap 1.11.x zu installieren.
 - Führen Sie den folgenden Befehl aus, um den AWS IoT Greengrass Snap zu installieren:

```
sudo snap install aws-iot-greengrass
```

Beispielantwort:

```
aws-iot-greengrass 1.11.5 from Amazon Web Services (aws) installed
```

- Führen Sie den folgenden Befehl aus, um von einer früheren Version auf v1.11.x zu migrieren oder auf die neueste verfügbare Patch-Version zu aktualisieren:

```
sudo snap refresh --channel=1.11.x aws-iot-greengrass
```

Wie andere Snaps verwendet der AWS IoT Greengrass Snap Kanäle, um Nebenversionen zu verwalten. Snaps werden automatisch auf die neueste verfügbare Version des aktuellen Kanals aktualisiert. Wenn Sie beispielsweise angeben `--channel=1.11.x`, wird Ihr AWS IoT Greengrass Snap auf Version 1.11.5 aktualisiert.

Sie können den `snap info aws-iot-greengrass` Befehl ausführen, um die Liste der verfügbaren Kanäle für abzurufen. AWS IoT Greengrass

Beispielantwort:

```
name:      aws-iot-greengrass
summary:   AWS supported software that extends cloud capabilities to local devices.
publisher: Amazon Web Services (aws#)
store-url: https://snapcraft.io/aws-iot-greengrass
contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
license:   Proprietary
description: |
  AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
  locally on the data
  they generate, while still using the cloud for management, analytics, and durable
  storage.
  AWS IoT Greengrass snap v1.11.0 enables you to run a limited version of AWS IoT
  Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
  inference.
  By downloading this software you agree to the Greengrass Core Software License
  Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
  license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
```

```
(https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
ggc.html#gg-snap-support) in
the AWS IoT Greengrass Developer.
If you need help, try the AWS IoT Greengrass tag on AWS re:Post
(https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
with an AWS IQ expert
(https://iq.aws.amazon.com/services/aws/greengrass).
snap-id: SRDuhPJGj4XPxFNNZQK0TvURAp0wxKnd
channels:
  latest/stable:    1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
  latest/beta:      1.11.3 2021-06-14 (59) 111MB -
  latest/edge:      1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. Um auf bestimmte Ressourcen zuzugreifen, die Ihre Lambda-Funktionen benötigen, können Sie eine Verbindung zu zusätzlichen Schnittstellen herstellen.

Führen Sie den folgenden Befehl aus, um die Liste der von AWS IoT Greengrass Snap unterstützten Schnittstellen abzurufen:

```
snap connections aws-iot-greengrass
```

Beispielantwort:

Interface	Notes	Plug	Slot
camera	-	aws-iot-greengrass:camera	-
dvb	-	aws-iot-greengrass:dvb	-
gpio	-	aws-iot-greengrass:gpio	-
gpio-memory-control	-	aws-iot-greengrass:gpio-memory-control	-
greengrass-support	-	aws-iot-greengrass:greengrass-support-no-container	-
:greengrass-support	-		
hardware-observe	-	aws-iot-greengrass:hardware-observe	-
:hardware-observe	-	manual	-

hardware-random-control	aws-iot-greengrass:hardware-random-control	-
-		
home	aws-iot-greengrass:home-for-greengrassd	-
-		
home	aws-iot-greengrass:home-for-hooks	:home
manual		
hugepages-control	aws-iot-greengrass:hugepages-control	
:hugepages-control	manual	
i2c	aws-iot-greengrass:i2c	-
-		
iio	aws-iot-greengrass:iio	-
-		
joystick	aws-iot-greengrass:joystick	-
-		
log-observe	aws-iot-greengrass:log-observe	:log-
observe	manual	
mount-observe	aws-iot-greengrass:mount-observe	
:mount-observe	manual	
network	aws-iot-greengrass:network	
:network	-	
network-bind	aws-iot-greengrass:network-bind	
:network-bind	-	
network-control	aws-iot-greengrass:network-control	
:network-control	-	
opengl	aws-iot-greengrass:opengl	
:opengl	-	
optical-drive	aws-iot-greengrass:optical-drive	
:optical-drive	-	
process-control	aws-iot-greengrass:process-control	
:process-control	-	
raw-usb	aws-iot-greengrass:raw-usb	-
-		
removable-media	aws-iot-greengrass:removable-media	-
-		
serial-port	aws-iot-greengrass:serial-port	-
-		
spi	aws-iot-greengrass:spi	-
-		
system-observe	aws-iot-greengrass:system-observe	
:system-observe	-	

Wenn Sie in der Spalte Slot einen Bindestrich (-) sehen, ist die entsprechende Schnittstelle nicht verbunden.

6. Folgen Sie [der Installation der AWS IoT Greengrass Core-Software](#), um ein AWS IoT Ding, eine Greengrass-Gruppe, Sicherheitsressourcen, die eine sichere Kommunikation mit ermöglichen AWS IoT, und der AWS IoT Greengrass Core-Software-Konfigurationsdatei zu erstellen. Die Konfigurationsdatei, `config.json`, enthält Konfigurationen, die für Ihren Greengrass-Kern spezifisch sind, z. B. den Speicherort der Zertifikatsdateien und den AWS IoT Gerätedatenendpunkt.

 Note

Wenn Sie die Datei auf ein anderes Gerät heruntergeladen haben, gehen Sie wie folgt vor, um die Dateien auf das AWS IoT Greengrass Kerngerät zu übertragen.

7. Stellen Sie für den AWS IoT Greengrass Snap sicher, dass Sie die Datei [config.json](#) aktualisieren, wie im Folgenden gezeigt:
 - Ersetzen Sie jede Instanz von `certificateId` durch die Zertifikat-ID im Namen des Zertifikats und der Schlüsseldateien.
 - Wenn Sie ein anderes Amazon-Root-CA-Zertifikat als Amazon Root CA 1 heruntergeladen haben, ersetzen Sie jede Instanz von `AmazonRootCA1.pem` durch den Namen der Amazon-Root-CA-Datei.

```
{
  ...
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key",
        "certificatePath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-certificate.pem.crt"
      }
    },
    "caPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/AmazonRootCA1.pem"
  },
}
```

```
"writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",  
"pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"  
}
```

8. Führen Sie den folgenden Befehl aus, um Ihr AWS IoT Greengrass Zertifikat und Ihre Konfigurationsdateien hinzuzufügen:

```
sudo snap set aws-iot-greengrass gg-certs=/home/ubuntu/my-certs
```

Bereitstellung einer Lambda-Funktion

In diesem Abschnitt erfahren Sie, wie Sie eine vom Kunden verwaltete Lambda-Funktion im AWS IoT Greengrass Handumdrehen bereitstellen.

Important

AWS IoT Greengrass Snap v1.11 unterstützt nur nicht containerisierte Lambda-Funktionen.

1. Führen Sie den folgenden Befehl aus, um den Daemon zu starten: AWS IoT Greengrass

```
sudo snap start aws-iot-greengrass
```

Beispielantwort:

```
Started.
```

Note

Wenn Sie eine Fehlermeldung erhalten, können Sie den `snap run` Befehl für eine detaillierte Fehlermeldung verwenden. Weitere Informationen zur Problembehandlung finden Sie unter [Fehler: Die folgenden Aufgaben können nicht ausgeführt werden: - Führen Sie den Dienstbefehl „start“ für die Dienste \[“greengrassd”\] von Snap "" aus \(\[Start Snap. aws-iot-greengrass aws-iot-greengrass.greengrassd.service\] ist mit dem Exit-Status 1 fehlgeschlagen: Job für Snap. aws-iot-greengrass.greengrassd.service ist fehlgeschlagen, weil der Kontrollprozess mit einem Fehlercode beendet wurde.](#)

Siehe „[Systemctl Status Snap](#)“. [aws-iot-greengrass](#)Einzelheiten finden Sie unter „[greengrassd.service](#)“ und „[journalctl -xe](#)“).

2. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob der Daemon ausgeführt wird:

```
snap services aws-iot-greengrass.greengrassd
```

Beispielantwort:

Service	Startup	Current	Notes
aws-iot-greengrass.greengrassd	disabled	active	-

3. Folgen Sie [Modul 3 \(Teil 1\): Lambda-Funktionen weiter AWS IoT Greengrass](#), um eine Hello World Lambda-Funktion zu erstellen und bereitzustellen. Bevor Sie jedoch die Lambda-Funktion bereitstellen, müssen Sie den nächsten Schritt abschließen.
4. Stellen Sie sicher, dass Ihre Lambda-Funktion als `snap_daemon` Benutzer und im Modus ohne Container ausgeführt wird. Gehen Sie in der AWS IoT Greengrass Konsole wie folgt vor, um die Einstellungen Ihrer Greengrass-Gruppe zu aktualisieren:
 - a. Melden Sie sich bei der AWS IoT Greengrass Konsole an.
 - b. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
 - c. Wählen Sie unter Greengrass-Gruppen die Zielgruppe aus.
 - d. Wählen Sie auf der Gruppenkonfigurationsseite im Navigationsbereich die Registerkarte Lambda-Funktionen aus.
 - e. Wählen Sie unter Standard-Laufzeitumgebung für Lambda-Funktionen die Option Bearbeiten aus, und gehen Sie wie folgt vor:
 - i. Wählen Sie für Standardsystembenutzer und -gruppe die Option Andere ID/group Benutzer-ID aus, und geben Sie **584788** dann sowohl Systembenutzer-ID (Nummer) als auch Systemgruppen-ID (Nummer) ein.
 - ii. Wählen Sie für Containerisierung der Standard-Lambda-Funktion die Option Kein Container aus.
 - iii. Wählen Sie Speichern.

Den Daemon stoppen AWS IoT Greengrass

Sie können den `snap stop` Befehl verwenden, um einen Dienst zu beenden.

Führen Sie den folgenden Befehl aus, um den AWS IoT Greengrass Daemon zu beenden:

```
sudo snap stop aws-iot-greengrass
```

Der Befehl sollte zurückkehren `Stopped..`

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob Sie den Snap erfolgreich gestoppt haben:

```
snap services aws-iot-greengrass.greengrassd
```

Beispielantwort:

Service	Startup	Current	Notes
aws-iot-greengrass.greengrassd	disabled	inactive	-

Den Snap deinstallieren AWS IoT Greengrass

Führen Sie den folgenden Befehl aus, um den AWS IoT Greengrass Snap zu deinstallieren:

```
sudo snap remove aws-iot-greengrass
```

Beispielantwort:

```
aws-iot-greengrass removed
```

Problembehandlung beim AWS IoT Greengrass Snap

Verwenden Sie die folgenden Informationen, um Probleme mit dem AWS IoT Greengrass Snap zu beheben.

Es wurden Fehler „Zugriff verweigert“ angezeigt.

Lösung: Fehler mit verweigerter Berechtigung sind häufig auf fehlende Schnittstellen zurückzuführen. Für eine Liste der fehlenden Schnittstellen und detaillierte Informationen zur Fehlerbehebung können Sie das `snappy-debug` Tool verwenden.

Führen Sie den folgenden Befehl aus, um das Tool zu installieren.

```
sudo snap install snappy-debug
```

Beispielantwort:

```
snappy-debug 0.36-snapd2.45.1 from Canonical# installed
```

Führen Sie den `sudo snappy-debug` Befehl in einer separaten Terminalsitzung aus. Der Vorgang wird fortgesetzt, bis der Fehler „Zugriff verweigert“ auftritt.

Wenn Ihre Lambda-Funktion beispielsweise versucht, eine Datei im `$HOME` Verzeichnis zu lesen, erhalten Sie möglicherweise die folgende Antwort:

```
INFO: Following '/var/log/syslog'. If have dropped messages, use:
INFO: $ sudo journalctl --output=short --follow --all | sudo snappy-debug
kernel.printk_ratelimit = 0
= AppArmor =
Time: Dec 6 04:48:26
Log: apparmor="DENIED" operation="mknod" profile="snap.aws-iot-greengrass.greengrassd"
     name="/home/ubuntu/my-file.txt" pid=12345 comm="touch" requested_mask="c"
     denied_mask="c" fsuid=0 ouid=0
File: /home/ubuntu/my-file.txt (write)
Suggestion:
* add 'home' to 'plugins'
```

Dieses Beispiel zeigt, dass das Erstellen der `/home/ubuntu/my-file.txt` Datei den Berechtigungsfehler verursacht hat. Es wird auch vorgeschlagen, dass Sie `home` zu `plugins` hinzufügen. Dieser Vorschlag ist jedoch nicht zutreffend. Die `home-for-hooks` Stecker `home-for-greengrassd` und haben nur Lesezugriff.

Weitere Informationen finden Sie unter [Der Snappy-Debug-Snap](#) in der Snap-Dokumentation.

Fehler: Die folgenden Aufgaben können nicht ausgeführt werden: - Führen Sie den Dienstbefehl „start“ für die Dienste [„greengrassd“] von Snap "" aus ([Start Snap. aws-iot-greengrass aws-iot-greengrass.greengrassd.service] ist mit dem Exit-Status 1 fehlgeschlagen: Job für Snap. aws-iot-greengrass.greengrassd.service ist fehlgeschlagen, weil der Kontrollprozess mit einem Fehlercode beendet wurde. Siehe „Systemctl Status Snap“. aws-iot-greengrassEinzelheiten finden Sie unter „greengrassd.service“ und „journalctl -xe“.)

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn der Befehl die snap start aws-iot-greengrass Core-Software nicht starten kann. AWS IoT Greengrass

Führen Sie den folgenden Befehl aus, um weitere Informationen zur Problembehandlung zu erhalten:

```
sudo snap run aws-iot-greengrass.greengrassd
```

Beispielantwort:

```
Couldn't find /snap/aws-iot-greengrass/44/greengrass/config/config.json.
```

Dieses Beispiel zeigt, dass die config.json Datei nicht gefunden werden AWS IoT Greengrass konnte. Sie könnten die Konfiguration und die Zertifikatsdateien überprüfen.

/var/snap/aws-iot-greengrass/current/ggc-write-directory/packages/1.11.5/rootfs/mergedist kein absoluter Pfad oder ein Symlink.

Lösung: Der AWS IoT Greengrass Snap unterstützt nur nicht containerisierte Lambda-Funktionen. Stellen Sie sicher, dass Sie Ihre Lambda-Funktionen im Modus ohne Container ausführen. Weitere Informationen finden Sie unter [Überlegungen zur Auswahl der Lambda-Funktionscontainerisierung im AWS IoT Greengrass Version 1 Entwicklerhandbuch](#).

Der snapd-Daemon konnte nicht neu gestartet werden, nachdem Sie den Befehl sudo snap refresh snapd ausgeführt haben.

Lösung: Folgen Sie den Schritten 6 bis 8 unter [Installieren und konfigurieren Sie den Snap AWS IoT Greengrass](#), um das AWS IoT Greengrass Zertifikat und die Konfigurationsdateien zum Snap hinzuzufügen. AWS IoT Greengrass

Archivieren Sie eine AWS IoT Greengrass Core-Software-Installation

Wenn Sie auf eine neue Version der AWS IoT Greengrass Core-Software aktualisieren, können Sie die aktuell installierte Version archivieren. Dadurch bleibt Ihre aktuelle Installationsumgebung

erhalten, sodass Sie eine neue Softwareversion auf derselben Hardware testen können. Dadurch wird auch ein Rollback auf Ihre archivierte Version vereinfacht.

So archivieren Sie die aktuelle Installation und installieren eine neue Version

1. Laden Sie das Installationspaket für die [AWS IoT Greengrass Core-Software](#) herunter, auf die Sie upgraden möchten.
2. Kopieren Sie das Paket in das Core-Zielgerät. Anweisungen zum Übertragen von Dateien finden Sie in diesem [Schritt](#).

 Note

Sie kopieren Ihre aktuellen Zertifikate, Schlüssel und die Konfigurationsdatei später in die neue Installation.

Führen Sie die Befehle in den folgenden Schritten in Ihrem Core-Geräte-Terminal aus.

3. Stellen Sie sicher, dass der Greengrass-Daemon auf dem Code-Gerät beendet wird.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/ggc-version/bin/daemon` enthält, dann wird der Daemon ausgeführt.

 Note

Dieses Verfahren wurde unter der Annahme geschrieben, dass die AWS IoT Greengrass Core-Software im `/greengrass` Verzeichnis installiert ist.

- b. So beenden Sie den -Daemon:

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

4. Verschieben Sie das aktuelle Greengrass-Stammverzeichnis in ein anderes Verzeichnis.

```
sudo mv /greengrass /greengrass_backup
```

5. Entpacken Sie die neue Software auf dem Core-Gerät. Ersetzen Sie die *version* Platzhalter *os-architecture* und im Befehl.

```
sudo tar -zxvf greengrass-os-architecture-version.tar.gz -C /
```

6. Kopieren Sie die archivierten Zertifikate, Schlüssel und die Konfigurationsdatei in die neue Installation.

```
sudo cp /greengrass_backup/certs/* /greengrass/certs  
sudo cp /greengrass_backup/config/* /greengrass/config
```

7. Starten Sie den Daemon:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

Jetzt können Sie eine Gruppenbereitstellung vornehmen, um die neue Installation zu testen. Bei einem Fehler können Sie die archivierte Installation wiederherstellen.

So stellen Sie die archivierte Installation wieder her

1. Beenden Sie den Daemon.
2. Löschen Sie das neue /greengrass-Verzeichnis.
3. Verschieben Sie das Verzeichnis /greengrass_backup wieder zu /greengrass.
4. Starten Sie den -Daemon.

Den AWS IoT Greengrass Kern konfigurieren

Ein AWS IoT Greengrass Kern ist ein AWS IoT Ding (Gerät), das in Edge-Umgebungen als Hub oder Gateway fungiert. Wie andere AWS IoT Geräte ist auch ein Core in der Registrierung vorhanden, hat einen Geräteshadow und verwendet ein Gerätezertifikat, um sich mit AWS IoT Core und AWS IoT Greengrass zu authentifizieren. Das Core-Gerät führt die AWS IoT Greengrass -Core-Software aus. So kann es die lokalen Prozesse für Greengrass-Gruppen verwalten, z. B. Kommunikation, Schattensynchronisierung und Austausch des Tokens.

Die AWS IoT Greengrass Core-Software bietet die folgenden Funktionen:

- Bereitstellung und lokale Ausführung von Konnektoren und Lambda-Funktionen.
- Verarbeiten Sie Datenströme lokal mit automatischen Exporten in die AWS Cloud.
- MQTT-Messaging über das lokale Netzwerk zwischen Geräten, Konnektoren und Lambda-Funktionen mithilfe verwalteter Abonnements.
- MQTT-Messaging zwischen AWS IoT Geräten, Konnektoren und Lambda-Funktionen mithilfe verwalteter Abonnements.
- Sichere Verbindungen zwischen Geräten und AWS Cloud Nutzung der Geräteauthentifizierung und -autorisierung.
- Lokale Shadow-Synchronisierung von Geräten. Shadows können so konfiguriert werden, dass sie mit dem synchronisiert werden AWS Cloud.
- Kontrollierter Zugriff auf lokale Geräte- und Volume-Ressourcen.
- Bereitstellung von Modellen für das maschinelle Lernen, die in der Cloud geschult werden, für die Ausführung lokaler Inferenzen.
- Automatische Erkennung von IP-Adressen, die Geräte in die Lage versetzt, das Greengrass Core-Gerät zu erkennen.
- Zentrale Bereitstellung neuer oder aktualisierter Gruppenkonfigurationen. Nach dem Download der Konfigurationsdaten wird das Core-Gerät automatisch neu gestartet.
- Sichere over-the-air (OTA) Softwareupdates von benutzerdefinierten Lambda-Funktionen.
- Sichere, verschlüsselte Speicherung lokaler Geheimnisse und kontrollierter Zugriff durch Konnektoren und Lambda-Funktionen.

AWS IoT Greengrass Kernkonfigurationsdatei

Die Konfigurationsdatei für die AWS IoT Greengrass Core-Software ist `config.json`. Sie befindet sich im Verzeichnis `/greengrass-root/config`.

Note

`greengrass-root` steht für den Pfad, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

Wenn Sie die Option Standardgruppenerstellung von der AWS IoT Greengrass Konsole aus verwenden, wird die `config.json` Datei in einem funktionierenden Zustand auf dem Core-Gerät bereitgestellt.

Sie können den Inhalt dieser Datei mit folgendem Befehl überprüfen:

```
cat /greengrass-root/config/config.json
```

Im Folgenden sehen Sie ein Beispiel für eine `config.json`-Datei. Dies ist die Version, die generiert wird, wenn Sie den Core von der AWS IoT Greengrass Konsole aus erstellen.

GGC v1.11

```
{
  "coreThing": {
    "caPath": "root.ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    "thingArn": "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600,
    "ggDaemonPort": 8000,
    "systemComponentAuthTimeout": 5000
  },
  "runtime": {
    "maxWorkItemCount": 1024,
    "maxConcurrentLimit": 25,
    "lruSize": 25,
    "mountAllBlockDevices": "no",
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
  "crypto": {
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate": {
```

```

        "privateKeyPath": "file:///greengrass/certs/hash.private.key",
        "certificatePath": "file:///greengrass/certs/hash.cert.pem"
    },
    "caPath": "file:///greengrass/certs/root.ca.pem"
},
"writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
"pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}

```

Die `config.json`-Datei unterstützt die folgenden Eigenschaften:

coreThing

Feld	Description	Hinweise
caPath	Der Pfad zur AWS IoT Stammzertifizierungsstelle relativ zum <code>/greengrass-root/certs</code> Verzeichnis.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das <code>crypto</code> Objekt vorhanden ist.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
certPath	Der Pfad zum Core-Gerätezertifikat, relativ zum Verzeichnis <code>/greengrass-root/certs</code> .	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das <code>crypto</code> Objekt vorhanden ist.
keyPath	Der Pfad zum privaten Core-Schlüssel, relativ zum Verzeichnis <code>/greengrass-root/certs</code> .	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird

Feld	Description	Hinweise
		ignoriert, wenn das crypto Objekt vorhanden ist.
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass get-core-definition-version CLI-Befehl ausführen.

Feld	Description	Hinweise
iotHost	Ihr AWS IoT Endpunkt.	Suchen Sie den Endpunkt in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI-Befehl ausführen. Dieser Befehl gibt den Amazon Trust Services (ATS) Endpunkt zurück. Weitere Informationen finden Sie in der Dokumentation Server-Authentifizierung.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region

Feld	Description	Hinweise
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Das ist Ihr <code>iotHost</code>-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. <code>greengrass-ats.iot . region.amazonaws.com</code>). Verwenden Sie dasselbe AWS-Region wie <code>iotHost</code>.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region
iotMqttPort	Optional. Die Portnummer, mit AWS IoT der für die MQTT-Kommunikation verwendet werden soll.	Gültige Werte sind 8883 oder 443. Der Standardwert ist 8883. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .

Feld	Description	Hinweise
iotHttpPort	Optional. Die zum Erstellen von HTTPS-Verbindungen mit AWS IoT verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
ggMqttPort	Optional. Die Portnummer, die für die MQTT-Kommunikation über das lokale Netzwerk verwendet werden soll.	Gültige Werte sind 1024 bis 65535. Der Standardwert ist 8883. Weitere Informationen finden Sie unter the section called "MQTT-Port für lokales Messaging" .
ggHttpPort	Optional. Die zum Erstellen von HTTPS-Verbindungen mit dem AWS IoT Greengrass-Service verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
keepAlive	Optional. Der MQTT-Zeitraum KeepAlive in Sekunden.	Der gültige Bereich liegt zwischen 30 und 1200 Sekunden. Der Standardwert ist 600.
networkProxy	Optional. Ein Objekt, das einen Proxy-Server definiert, mit dem eine Verbindung hergestellt werden soll.	Der Proxyserver kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .

Feld	Description	Hinweise
<code>mqttOperationTimeout</code>	Optional. Die Zeit (in Sekunden), für die es dem Greengrass-Core möglich ist, einen Veröffentlichungs-, Abonnement- oder Abmeldevorgang in MQTT-Verbindungen zu AWS IoT Core abzuschließen.	Der Standardwert ist 5. Der Mindestwert ist 5.
<code>ggDaemonPort</code>	Optional. Die Greengrass-Kern-IPC-Portnummer.	Diese Eigenschaft ist in AWS IoT Greengrass Version 1.11.0 oder höher verfügbar. Gültige Werte liegen zwischen 1024 und 65535. Der Standardwert ist 8000.
<code>systemComponentAuthTimeout</code>	Optional. Die Zeit (in Millisekunden), bis der Greengrass-Core-IPC die Authentifizierung abschließen kann.	Diese Eigenschaft ist in Version 1.11.0 oder höher verfügbar. AWS IoT Greengrass Gültige Werte liegen zwischen 500 und 5000. Der Standardwert ist 5000.

runtime

Feld	Description	Hinweise
<code>maxWorkItemCount</code>	Optional. Die maximale Anzahl von Arbeitselementen, die der Greengrass-Daemon gleichzeitig verarbeiten kann.	Der Standardwert lautet 1024. Der Maximalwert wird durch Ihre Gerätehardware begrenzt.

Feld	Description	Hinweise
	Arbeitselemente, die diesen Grenzwert überschreiten, werden ignoriert. Die Warteschlange für Arbeitsaufgaben wird von Systemkomponenten, benutzerdefinierten Lambda-Funktionen und Konnektoren gemeinsam genutzt.	Wenn Sie diesen Wert erhöhen, erhöht sich der Speicherverbrauch. AWS IoT Greengrass Sie können diesen Wert erhöhen, wenn Sie erwarten, dass Ihr Core einen hohen MQTT-Nachrichtenverkehr empfängt.
<code>maxConcurrentLimit</code>	Optional. Die maximale Anzahl an gleichzeitigen Lambda-Workern ohne Pin, die der Greengrass-Daemon haben kann. Sie können eine andere Ganzzahl angeben, um diesen Parameter zu überschreiben.	Der Standardwert ist 25. Der Mindestwert ist definiert durch <code>lruSize</code>.
<code>lruSize</code>	Optional. Definiert den Mindestwert für <code>maxConcurrentLimit</code>.	Der Standardwert ist 25.
<code>mountAllBlockDevices</code>	Optional. Ermöglicht AWS IoT Greengrass die Verwendung von Bind-Mounts, um alle Blockgeräte nach der Einrichtung von OverlayFS in einen Container einzuhängen.	Diese Eigenschaft ist in AWS IoT Greengrass Version 1.11.0 oder höher verfügbar. Gültige Werte sind <code>yes</code> und <code>no</code>. Der Standardwert ist <code>no</code>. Setzen Sie diesen Wert auf <code>yes</code> wenn sich Ihr <code>/usr</code> Verzeichnis nicht in der <code>/</code> Hierarchie befindet.

Feld	Description	Hinweise
postStartHealthCheckTimeout	Optional. Die Zeit (in Millisekunden) nach dem Start, die der Greengrass-Daemon wartet, bis die Zustandsprüfung beendet ist.	Das Standard-Timeout beträgt 30 Sekunden (30.000 ms).
cgroup		
useSystemd	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind yes oder no. Führen Sie das <code>check_ggc_dependencies</code> -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät <code>systemd</code> verwendet.

crypto

Das `crypto`-Objekt enthält Eigenschaften, die die Speicherung privater Schlüssel auf einem Hardware-Sicherheitsmodul (HSM) über PKCS#11 und lokale geheime Speicher unterstützen. Weitere Informationen finden Sie unter [the section called "Sicherheitsprinzipale"](#), [the section called "Integration von Hardware-Sicherheit"](#) und [Bereitstellen von Secrets für den Core](#). Konfigurationen für die Speicherung privater Schlüssel auf HSMs oder im Dateisystem werden unterstützt.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .

Note

Stellen Sie sicher, dass Ihre [Endpunkte](#)

Feld	Description	Hinweise
		<u>Ihrem Zertifikatstyp entsprechen.</u>
PKCS11		
OpenSSLEngine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren.	Muss ein Pfad zu einer Datei im Dateisystem sein. Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardware-sicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates” .
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	

Feld	Description	Hinweise
<code>IoTCertificate.privateKeyPath</code>	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
<code>IoTCertificate.certificatePath</code>	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .
<code>MQTTServerCertificate</code>	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate.privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code>. Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Die folgenden Konfigurationseigenschaften werden ebenfalls unterstützt:

Feld	Description	Hinweise
mqttMaxConnectionRetryInterval	Optional. Das Maximalintervall (in Sekunden) zwischen MQTT-Verbindungsversuchen, wenn die Verbindung getrennt wird.	Geben Sie diesen Wert als Ganzzahl ohne Vorzeichen an. Der Standardwert ist 60.
managedRespawn	Optional. Gibt an, dass der OTA-Agent vor einem Update einen benutzerdefinierten Code ausführen muss.	Gültige Werte sind <code>true</code> oder <code>false</code> . Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

Feld	Description	Hinweise
writeDirectory	Optional. Das Schreibverzeichnis, in dem alle read/write Ressourcen AWS IoT Greengrass erstellt werden.	Weitere Informationen finden Sie unter Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass .
pidFileDirectory	Fakultativ. AWS IoT Greengrass speichert seine Prozess-ID (PID) in diesem Verzeichnis.	Der Standardwert ist /var/run.

Extended life versions

Die folgenden Versionen der AWS IoT Greengrass Core-Software befinden sich in der [erweiterten Lebensphase](#). Diese Informationen dienen lediglich Referenzzwecken.

GGC v1.10

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600,
    "systemComponentAuthTimeout": 5000
  },
  "runtime" : {
    "maxWorkItemCount" : 1024,
    "maxConcurrentLimit" : 25,
    "lruSize": 25,
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
```

```

    "SecretsManager" : {
      "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
    },
    "IoTCertificate" : {
      "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
      "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
    }
  },
  "caPath" : "file:///greengrass/certs/root.ca.pem"
}

```

Die config.json-Datei unterstützt die folgenden Eigenschaften:

coreThing

Feld	Description	Hinweise
caPath	Der Pfad zur AWS IoT Root-CA relativ zum <i>/greengrass-root /</i> certs Verzeichnis.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
certPath	Der Pfad zum Core-Gerätezertifikat, relativ zum Verzeichnis <i>/greengrass-root /</i> certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.

Feld	Description	Hinweise
keyPath	Der Pfad zum privaten Core-Schlüssel, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass get-core-definition-version CLI-Befehl ausführen.

Feld	Description	Hinweise
iotHost	Ihr AWS IoT Endpunkt.	Suchen Sie den Endpunkt in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI-Befehl ausführen. Dieser Befehl gibt den Amazon Trust Services (ATS) Endpunkt zurück. Weitere Informationen finden Sie in der Dokumentation Server-Authentifizierung.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region

Feld	Description	Hinweise
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Das ist Ihr <code>iotHost</code>-Endpunkt mit dem durch <code>greengrass</code> ersetzten Host-Präfix (z. B. <code>greengrass-ats.iot</code> . region .amazonaws.com). Verwenden Sie dasselbe AWS-Region wie <code>iotHost</code>.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region
iotMqttPort	Optional. Die Portnummer, mit AWS IoT der für die MQTT-Kommunikation verwendet werden soll.	Gültige Werte sind 8883 oder 443. Der Standardwert ist 8883. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Feld	Description	Hinweise
<code>iotHttpPort</code>	Optional. Die zum Erstellen von HTTPS-Verbindungen mit AWS IoT verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
<code>ggMqttPort</code>	Optional. Die Portnummer, die für die MQTT-Kommunikation über das lokale Netzwerk verwendet werden soll.	Gültige Werte sind 1024 bis 65535. Der Standardwert ist 8883. Weitere Informationen finden Sie unter the section called "MQTT-Port für lokales Messaging" .
<code>ggHttpPort</code>	Optional. Die zum Erstellen von HTTPS-Verbindungen mit dem AWS IoT Greengrass -Service verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
<code>keepAlive</code>	Optional. Der MQTT-Zeitraum <code>KeepAlive</code> in Sekunden.	Der gültige Bereich liegt zwischen 30 und 1200 Sekunden. Der Standardwert ist 600.
<code>networkProxy</code>	Optional. Ein Objekt, das einen Proxy-Server definiert, mit dem eine Verbindung hergestellt werden soll.	Der Proxyserver kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .

Feld	Description	Hinweise
<code>mqttOperationTimeout</code>	Optional. Die Zeit (in Sekunden), für die es dem Greengrass-Core möglich ist, einen Veröffentlichungs-, Abonnement- oder Abmeldevorgang in MQTT-Verbindungen zu AWS IoT Core abzuschließen.	Diese Eigenschaft ist ab AWS IoT Greengrass Version 1.10.2 verfügbar. Der Standardwert ist 5. Der Mindestwert ist 5.
<code>runtime</code>		

Feld	Description	Hinweise
<code>maxWorkItemCount</code>	Optional. Die maximale Anzahl von Arbeitselementen, die der Greengrass-Daemon gleichzeitig verarbeiten kann. Arbeitselemente, die diesen Grenzwert überschreiten, werden ignoriert. Die Warteschlange für Arbeitsaufgaben wird von Systemkomponenten, benutzerdefinierten Lambda-Funktionen und Konnektoren gemeinsam genutzt.	Der Standardwert lautet 1024. Der Maximalwert wird durch Ihre Gerätehardware begrenzt. Wenn Sie diesen Wert erhöhen, erhöht sich der Speicherverbrauch. AWS IoT Greengrass Sie können diesen Wert erhöhen, wenn Sie erwarten, dass Ihr Core einen hohen MQTT-Nachrichtenverkehr empfängt.
<code>maxConcurrentLimit</code>	Optional. Die maximale Anzahl an gleichzeitigen Lambda-Workern ohne Pin, die der Greengrass	Der Standardwert ist 25. Der Mindestwert ist definiert durch <code>lruSize</code> .

Feld	Description	Hinweise
	s-Daemon haben kann. Sie können eine andere Ganzzahl angeben, um diesen Parameter zu überschreiben.	
<code>lruSize</code>	Optional. Definiert den Mindestwert für <code>maxConcurrentLimit</code> .	Der Standardwert ist 25.
<code>postStartHealthCheckTimeout</code>	Optional. Die Zeit (in Millisekunden) nach dem Start, die der Greengrass-Daemon wartet, bis die Zustandsprüfung beendet ist.	Das Standard-Timeout beträgt 30 Sekunden (30.000 ms).
<code>cgroup</code>		
<code>useSystemd</code>	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind <code>yes</code> oder <code>no</code> . Führen Sie das <code>check_ggc_dependencies</code> -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät <code>systemd</code> verwendet.

crypto

Das `crypto`-Objekt enthält Eigenschaften, die die Speicherung privater Schlüssel auf einem Hardware-Sicherheitsmodul (HSM) über PKCS#11 und lokale geheime Speicher unterstützen. Weitere Informationen finden Sie unter [the section called "Sicherheitsprinzipale"](#), [the section called "Integration von Hardware-Sicherheit"](#) und [Bereitstellen von Secrets für den Core](#). Konfigurationen für die Speicherung privater Schlüssel auf HSMs oder im Dateisystem werden unterstützt.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
PKCS11		
OpenSSLEngine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren.	Muss ein Pfad zu einer Datei im Dateisystem sein. Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardware-Sicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates”.
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.

Feld	Description	Hinweise
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Die folgenden Konfigurationseigenschaften werden ebenfalls unterstützt:

Feld	Description	Hinweise
mqttMaxConnectionRetryInterval	Optional. Das Maximalintervall (in Sekunden) zwischen MQTT-Verbindungsversuchen, wenn die Verbindung getrennt wird.	Geben Sie diesen Wert als Ganzzahl ohne Vorzeichen an. Der Standardwert ist 60.
managedRespawn	Optional. Gibt an, dass der OTA-Agent vor einem Update einen benutzerdefinierten Code ausführen muss.	Gültige Werte sind <code>true</code> oder <code>false</code> . Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

Feld	Description	Hinweise
writeDirectory	Optional. Das Schreibverzeichnis, in dem alle read/write Ressourcen AWS IoT Greengrass erstellt werden.	Weitere Informationen finden Sie unter Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass .

GGC v1.9

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

Die config.json-Datei unterstützt die folgenden Eigenschaften:

coreThing

Feld	Description	Hinweise
caPath	Der Pfad zur AWS IoT Root-CA relativ zum <i>/greengrass-root</i> /certs Verzeichnis.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
certPath	Der Pfad zum Core-Gerätezertifikat, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
keyPath	Der Pfad zum privaten Core-Schlüssel, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass

Feld	Description	Hinweise
		get-core-definition-version CLI-Befehl ausführen.
iotHost	Ihr AWS IoT Endpunkt.	Suchen Sie den Endpunkt in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI-Befehl ausführen. Dieser Befehl gibt den Amazon Trust Services (ATS) Endpunkt zurück. Weitere Informationen finden Sie in der Dokumentation Server-Authentifizierung.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region

Feld	Description	Hinweise
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Das ist Ihr <code>iotHost</code>-Endpunkt mit dem durch <code>greengrass</code> ersetzten Host-Präfix (z. B. <code>greengrass-ats.iot</code> . region .amazonaws.com). Verwenden Sie dasselbe AWS-Region wie <code>iotHost</code>.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region
iotMqttPort	Optional. Die Portnummer, mit AWS IoT der für die MQTT-Kommunikation verwendet werden soll.	Gültige Werte sind 8883 oder 443. Der Standardwert ist 8883. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Feld	Description	Hinweise
<code>iotHttpPort</code>	Optional. Die zum Erstellen von HTTPS-Verbindungen mit AWS IoT verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
<code>ggHttpPort</code>	Optional. Die zum Erstellen von HTTPS-Verbindungen mit dem AWS IoT Greengrass -Service verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
<code>keepAlive</code>	Optional. Der MQTT-Zeitraum KeepAlive in Sekunden.	Der gültige Bereich liegt zwischen 30 und 1200 Sekunden. Der Standardwert ist 600.
<code>networkProxy</code>	Optional. Ein Objekt, das einen Proxy-Server definiert, mit dem eine Verbindung hergestellt werden soll.	Der Proxyserver kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .

runtime

Feld	Description	Hinweise
<code>maxConcurrentLimit</code>	Optional. Die maximale Anzahl an gleichzeitigen Lambda-Workern ohne	Der Standardwert ist 25. Der Mindestwert ist definiert durch <code>lruSize</code> .

Feld	Description	Hinweise
	Pin, die der Greengrass-Daemon haben kann. Sie können eine andere Ganzzahl angeben, um diesen Parameter zu überschreiben.	
<code>lruSize</code>	Optional. Definiert den Mindestwert für <code>maxConcurrentLimit</code> .	Der Standardwert ist 25.
<code>postStartHealthCheckTimeout</code>	Optional. Die Zeit (in Millisekunden) nach dem Start, die der Greengrass-Daemon wartet, bis die Zustandsprüfung beendet ist.	Das Standard-Timeout beträgt 30 Sekunden (30.000 ms).
<code>cgroup</code>		
<code>useSystemd</code>	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind <code>yes</code> oder <code>no</code> . Führen Sie das <code>check_ggc_dependencies</code> -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät <code>systemd</code> verwendet.

crypto

Das `crypto`-Objekt wurde in v1.7.0 hinzugefügt. Es führt Eigenschaften ein, die die Speicherung privater Schlüssel auf einem Hardware-Sicherheitsmodul (HSM) über PKCS#11 und lokale geheime Speicher unterstützen. Weitere Informationen finden Sie unter [the section called "Sicherheitsprinzipale"](#), [the section called "Integration von Hardware-Sicherheit"](#) und [Bereitstellen von Secrets für den Core](#). Konfigurationen für die Speicherung privater Schlüssel auf HSMs oder im Dateisystem werden unterstützt.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
PKCS11		
OpenSSLEngine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren.	Muss ein Pfad zu einer Datei im Dateisystem sein. Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardware-Sicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates” .
P11Provider	Der absolute Pfad zur ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.

Feld	Description	Hinweise
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Die folgenden Konfigurationseigenschaften werden ebenfalls unterstützt.

Feld	Description	Hinweise
mqttMaxConnectionRetryInterval	Optional. Das Maximalintervall (in Sekunden) zwischen MQTT-Verbindungsversuchen, wenn die Verbindung getrennt wird.	Geben Sie diesen Wert als Ganzzahl ohne Vorzeichen an. Der Standardwert ist 60.
managedRespawn	Optional. Gibt an, dass der OTA-Agent vor einem Update einen benutzerdefinierten Code ausführen muss.	Gültige Werte sind <code>true</code> oder <code>false</code> . Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

Feld	Description	Hinweise
writeDirectory	Optional. Das Schreibverzeichnis, in dem alle read/write Ressourcen AWS IoT Greengrass erstellt werden.	Weitere Informationen finden Sie unter Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass .

GGC v1.8

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

Die config.json Datei unterstützt die folgenden Eigenschaften.

coreThing

Feld	Description	Hinweise
caPath	Der Pfad zur AWS IoT Stammzertifizierungsstelle relativ zum <i>/greengrass-root</i> /certs Verzeichnis.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
certPath	Der Pfad zum Core-Gerätezertifikat, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
keyPath	Der Pfad zum privaten Core-Schlüssel, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass

Feld	Description	Hinweise
		get-core-definition-version CLI-Befehl ausführen.
iotHost	Ihr AWS IoT Endpunkt.	Suchen Sie den Endpunkt in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI-Befehl ausführen. Dieser Befehl gibt den Amazon Trust Services (ATS) Endpunkt zurück. Weitere Informationen finden Sie in der Dokumentation Server-Authentifizierung.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region

Feld	Description	Hinweise
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Das ist Ihr <code>iotHost</code>-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. <code>greengrass-ats.iot</code> . region .amazonaws.com). Verwenden Sie dasselbe AWS-Region wie <code>iotHost</code>.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region
iotMqttPort	Optional. Die Portnummer, mit der Sie für die MQTT-Kommunikation verwenden möchten. AWS IoT	Gültige Werte sind 8883 oder 443. Der Standardwert ist 8883. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy.

Feld	Description	Hinweise
iotHttpPort	Optional. Die zum Erstellen von HTTPS-Verbindungen mit AWS IoT verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
ggHttpPort	Optional. Die zum Erstellen von HTTPS-Verbindungen mit dem AWS IoT Greengrass -Service verwendete Portnummer.	Gültige Werte sind 8443 oder 443. Der Standardwert ist 8443. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
keepAlive	Optional. Der MQTT-Zeitraum KeepAlive in Sekunden.	Der gültige Bereich liegt zwischen 30 und 1200 Sekunden. Der Standardwert ist 600.
networkProxy	Optional. Ein Objekt, das einen Proxy-Server definiert, mit dem eine Verbindung hergestellt werden soll.	Der Proxyserver kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .

runtime

Feld

Description

Hinweise

cgroup

Feld	Description	Hinweise
useSystemd	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind yes oder no. Führen Sie das <code>check_ggc_dependencies</code> -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät systemd verwendet.

crypto

Das `crypto`-Objekt wurde in v1.7.0 hinzugefügt. Es führt Eigenschaften ein, die die Speicherung privater Schlüssel auf einem Hardware-Sicherheitsmodul (HSM) über PKCS#11 und lokale geheime Speicher unterstützen. Weitere Informationen finden Sie unter [the section called “Sicherheitsprinzipale”](#), [the section called “Integration von Hardware-Sicherheit”](#) und [Bereitstellen von Secrets für den Core](#). Konfigurationen für die Speicherung privater Schlüssel auf HSMs oder im Dateisystem werden unterstützt.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .

Note

Stellen Sie sicher, dass Ihre [Endpunkte Ihrem Zertifikatstyp entsprechen](#).

PKCS11		
OpenSSL Engine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-U	Muss ein Pfad zu einer Datei im Dateisystem sein.

Feld	Description	Hinweise
	Unterstützung unter OpenSSL zu aktivieren.	Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardware-Sicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates” .
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	

Feld	Description	Hinweise
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate.privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Die folgenden Konfigurationseigenschaften werden ebenfalls unterstützt:

Feld	Description	Hinweise
mqttMaxConnectionRetryInterval	Optional. Das Maximalintervall (in Sekunden) zwischen MQTT-Verbindungsversuchen, wenn die Verbindung getrennt wird.	Geben Sie diesen Wert als Ganzzahl ohne Vorzeichen an. Der Standardwert ist 60.
managedRespawn	Optional. Gibt an, dass der OTA-Agent vor einem Update einen benutzerdefinierten Code ausführen muss.	Gültige Werte sind <code>true</code> oder <code>false</code> . Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

Feld	Description	Hinweise
writeDirectory	Optional. Das Schreibverzeichnis, in dem alle read/write Ressourcen AWS IoT Greengrass erstellt werden.	Weitere Informationen finden Sie unter Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass .

GGC v1.7

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

Die config.json-Datei unterstützt die folgenden Eigenschaften:

coreThing

Feld	Description	Hinweise
caPath	Der Pfad zur AWS IoT Stammzertifizierungsstelle relativ zum <i>/greengrass-root</i> /certs Verzeichnis.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
certPath	Der Pfad zum Core-Gerätezertifikat, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
keyPath	Der Pfad zum privaten Core-Schlüssel, relativ zum Verzeichnis <i>/greengrass-root</i> /certs.	Aus Gründen der Abwärtskompatibilität mit Versionen vor 1.7.0. Diese Eigenschaft wird ignoriert, wenn das crypto Objekt vorhanden ist.
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass

Feld	Description	Hinweise
		get-core-definition-version CLI-Befehl ausführen.
iotHost	Ihr AWS IoT Endpunkt.	Suchen Sie den Endpunkt in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint --endpoint-type iot:Data-ATS CLI-Befehl ausführen. Dieser Befehl gibt den Amazon Trust Services (ATS) Endpunkt zurück. Weitere Informationen finden Sie in der Dokumentation Server-Authentifizierung.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region

Feld	Description	Hinweise
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Das ist Ihr <code>iotHost</code>-Endpunkt mit dem durch greengrass ersetzten Host-Präfix (z. B. <code>greengrass-ats.iot</code> . region .amazonaws.com). Verwenden Sie dasselbe AWS-Region wie <code>iotHost</code>.  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen. Stellen Sie sicher, dass Ihre Endpunkte Ihren entsprechen. AWS-Region
iotMqttPort	Optional. Die Portnummer, mit der Sie für die MQTT-Kommunikation verwenden möchten. AWS IoT	Gültige Werte sind 8883 oder 443. Der Standardwert ist 8883. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .
keepAlive	Optional. Der MQTT-Zeitraum <code>KeepAlive</code> in Sekunden.	Der gültige Bereich liegt zwischen 30 und 1200 Sekunden. Der Standardwert ist 600.

Feld	Description	Hinweise
networkProxy	Optional. Ein Objekt, das einen Proxy-Server definiert, mit dem eine Verbindung hergestellt werden soll.	Der Proxyserver kann HTTP oder HTTPS sein. Weitere Informationen finden Sie unter Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy .

runtime

Feld	Description	Hinweise
cgroup		
useSystemd	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind yes oder no. Führen Sie das <code>check_ggc_dependencies</code> -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät <code>systemd</code> verwendet.

crypto

Das in v1.7.0 hinzugefügte `crypto`-Objekt führt Eigenschaften ein, die die Speicherung privater Schlüssel auf einem Hardware-Sicherheitsmodul (HSM) über PKCS#11 und lokale geheime Speicher unterstützen. Weitere Informationen erhalten Sie unter [the section called "Integration von Hardware-Sicherheit"](#) und [Bereitstellen von Secrets für den Core](#). Konfigurationen für die Speicherung privater Schlüssel auf HSMs oder im Dateisystem werden unterstützt.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein:

Feld	Description	Hinweise
		<code>file:///absolute/path/to/file</code> .  Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.
PKCS11		
OpenSSL Engine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren.	Muss ein Pfad zu einer Datei im Dateisystem sein. Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardware-Sicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates”.
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.

Feld	Description	Hinweise
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Die folgenden Konfigurationseigenschaften werden ebenfalls unterstützt:

Feld	Description	Hinweise
mqttMaxConnectionRetryInterval	Optional. Das Maximalintervall (in Sekunden) zwischen MQTT-Verbindungsversuchen, wenn die Verbindung getrennt wird.	Geben Sie diesen Wert als Ganzzahl ohne Vorzeichen an. Der Standardwert ist 60.
managedRespawn	Optional. Gibt an, dass der OTA-Agent vor einem Update einen benutzerdefinierten Code ausführen muss.	Gültige Werte sind <code>true</code> oder <code>false</code> . Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

Feld	Description	Hinweise
<code>writeDirectory</code>	Optional. Das Schreibverzeichnis, in dem alle read/write Ressourcen AWS IoT Greengrass erstellt werden.	Weitere Informationen finden Sie unter Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass .

GGC v1.6

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600,
    "mqttMaxConnectionRetryInterval": 60
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true,
  "writeDirectory": "/write-directory"
}
```

 Note

Wenn Sie die Option zur Erstellung von Standardgruppen von der AWS IoT Greengrass Konsole aus verwenden, wird die `config.json` Datei in einem funktionierenden Zustand, der die Standardkonfiguration angibt, auf dem Kerngerät bereitgestellt.

Die `config.json`-Datei unterstützt die folgenden Eigenschaften:

Feld	Description	Hinweise
caPath	Der Pfad zum AWS IoT Root CA , relativ zum Verzeichnis <code>/greengrass-root/certs</code> .	Speichern Sie die Datei unter <code>/greengrass-root/certs</code> .
certPath	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum <code>/greengrass-root/certs</code> Verzeichnis.	Speichern Sie die Datei unter <code>/greengrass-root/certs</code> .
keyPath	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum <code>/greengrass-root/certs</code> Verzeichnis.	Speichern Sie die Datei unter <code>/greengrass-root/certs</code> .
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass get-core-definition-version CLI-Befehl ausführen.
iotHost	Ihr AWS IoT Endpunkt.	Finden Sie dies in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint CLI-Befehl ausführen.
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Für diesen Wert wird das <code>greengrass.s.iot.region.amazonaws</code>

Feld	Description	Hinweise
<code>keepAlive</code>	Der MQTT-Zeitraum <code>KeepAlive</code> in Sekunden.	<code>s.com</code> -Format verwendet. Verwenden Sie dieselbe Region wie <code>iotHost</code>. Dies ist ein optionaler Wert. Der Standardwert ist <code>600</code>.
<code>mqttMaxConnectionRetryInterval</code>	Das Maximalintervall (in Sekunden) zwischen MQTT-Verbindungsversuchen, wenn die Verbindung getrennt wird.	Geben Sie diesen Wert als Ganzzahl ohne Vorzeichen an. Dies ist ein optionaler Wert. Der Standardwert ist <code>60</code> .
<code>useSystemd</code>	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind <code>yes</code> oder <code>no</code> . Führen Sie das <code>check_ggc_dependencies</code> -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät <code>systemd</code> verwendet.
<code>managedRespawn</code>	Eine optionale over-the-air (OTA-) Aktualisierungsfunktion, die darauf hinweist, dass der OTA-Agent vor einem Update benutzerdefinierten Code ausführen muss.	Gültige Werte sind <code>true</code> oder <code>false</code> . Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .
<code>writeDirectory</code>	Das Schreibverzeichnis, in dem alle read/write Ressourcen AWS IoT Greengrass erstellt werden.	Dies ist ein optionaler Wert. Weitere Informationen finden Sie unter Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass .

GGC v1.5

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true
}
```

Die Datei `config.json` befindet sich in `/greengrass-root/config` und enthält die folgenden Parameter:

Feld	Description	Hinweise
caPath	Der Pfad zum AWS IoT Root CA , relativ zum Ordner <code>/greengrass-root/certs</code> .	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .
certPath	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum <code>/greengrass-root/certs</code> Ordner.	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .
keyPath	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum <code>/greengrass-root/certs</code> Ordner.	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .

Feld	Description	Hinweise
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das das AWS IoT Greengrass Kerngerät darstellt.	Suchen Sie den ARN für Ihren Core in der AWS IoT Greengrass Konsole unter Cores oder indem Sie den aws greengrass get-core-definition-version CLI-Befehl ausführen.
iotHost	Ihr AWS IoT Endpunkt.	Finden Sie dies in der AWS IoT Konsole unter Einstellungen oder indem Sie den aws iot describe-endpoint Befehl ausführen.
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Für diesen Wert wird das greengrass.s.iot. <i>region</i> .amazonaws.com -Format verwendet. Verwenden Sie dieselbe Region wie iotHost.
keepAlive	Der MQTT-Zeitraum KeepAlive in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.
useSystemd	Gibt an, ob Ihr Gerät systemd verwendet.	Gültige Werte sind yes oder no. Führen Sie das check_ggc_dependencies -Skript in Modul 1 aus, um festzustellen, ob Ihr Gerät systemd verwendet.

Feld	Description	Hinweise
managedRespawn	Eine optionale over-the-air (OTA-) Aktualisierungsfunktion, die darauf hinweist, dass der OTA-Agent vor einem Update benutzerdefinierten Code ausführen muss.	Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

GGC v1.3

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  },
  "managedRespawn": true
}
```

Die Datei config.json befindet sich in `/greengrass-root/config` und enthält die folgenden Parameter:

Feld	Description	Hinweise
caPath	Der Pfad zum AWS IoT Root CA , relativ zum Ordner	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .

Feld	Description	Hinweise
	<code>/greengrass-root / certs.</code>	
certPath	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum <code>/greengrass-root / certs</code> Ordner.	Speichern Sie die Datei im Ordner <code>/greengrass-root / certs</code> .
keyPath	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum <code>/greengrass-root / certs</code> Ordner.	Speichern Sie die Datei im Ordner <code>/greengrass-root / certs</code> .
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das den AWS IoT Greengrass Kern darstellt.	Sie finden diesen Wert in der AWS IoT Greengrass Konsole unter der Definition für Ihr AWS IoT Ding.
iotHost	Dein AWS IoT Endpunkt.	Sie finden diesen Wert in der AWS IoT Konsole unter Einstellungen.
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Sie finden diesen Wert in der AWS IoT Konsole unter Einstellungen mit vorangestelltem <code>greengrass.</code> Präfix.
keepAlive	Der MQTT-Zeitraum KeepAlive in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.

Feld	Description	Hinweise
useSystemd	Ein Binär-Marker, wenn Ihr Gerät systemd verwendet.	Die Werte sind yes oder no. Verwenden Sie das Abhängigkeitskript in Modul 1 , um festzustellen, ob Ihr Gerät systemd verwendet.
managedRespawn	Eine optionale over-the-air (OTA-) Aktualisierungsfunktion, die darauf hinweist, dass der OTA-Agent vor einem Update benutzerdefinierten Code ausführen muss.	Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software .

GGC v1.1

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  }
}
```

Die Datei `config.json` befindet sich in `/greengrass-root/config` und enthält die folgenden Parameter:

Feld	Description	Hinweise
caPath	Der Pfad zum AWS IoT Root CA , relativ zum Ordner <code>/greengrass-root / certs</code> .	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .
certPath	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum <code>/greengrass-root / certs</code> Ordner.	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .
keyPath	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum <code>/greengrass-root / certs</code> Ordner.	Speichern Sie die Datei im Ordner <code>/greengrass-root/certs</code> .
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das den AWS IoT Greengrass Kern darstellt.	Sie finden diesen Wert in der AWS IoT Greengrass Konsole unter der Definition für Ihr AWS IoT Ding.
iotHost	Dein AWS IoT Endpunkt.	Sie finden diesen Wert in der AWS IoT Konsole unter Einstellungen.
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Sie finden diesen Wert in der AWS IoT Konsole unter Einstellungen mit vorangestelltem <code>greengrass.</code> Präfix.
keepAlive	Der MQTT-Zeitraum <code>KeepAlive</code> in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.

Feld	Description	Hinweise
useSystemd	Ein Binär-Marker, wenn Ihr Gerät systemd verwendet.	Die Werte sind yes oder no. Verwenden Sie das Abhängigkeitskript in Modul 1 , um festzustellen, ob Ihr Gerät systemd verwendet.

GGC v1.0

In AWS IoT Greengrass Core v1.0 `config.json` wird es für bereitgestellt. *greengrass-root*/configuration

```
{
  "coreThing": {
    "caPath": "root-ca-pem",
    "certPath": "cloud-pem-crt",
    "keyPath": "cloud-pem-key",
    "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost": "host-prefix.iot.region.amazonaws.com",
    "ggHost": "greengrass.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes/no"
    }
  }
}
```

Die Datei `config.json` befindet sich in */greengrass-root*/configuration und enthält die folgenden Parameter:

Feld	Description	Hinweise
caPath	Der Pfad zum AWS IoT Root CA , relativ zum Ordner <i>/greengrass-root</i> /	Speichern Sie die Datei im Ordner <i>/greengrass-root</i>

Feld	Description	Hinweise
	configuration/certs .	<i>root</i> /configuration/certs .
certPath	Der Pfad zum AWS IoT Greengrass Kernzertifikat relativ zum <i>/greengrass-root</i> /configuration/certs Ordner.	Speichern Sie die Datei im Ordner <i>/greengrass-root</i> /configuration/certs .
keyPath	Der Pfad zum privaten AWS IoT Greengrass Kernschlüssel relativ zum <i>/greengrass-root</i> /configuration/certs Ordner.	Speichern Sie die Datei im Ordner <i>/greengrass-root</i> /configuration/certs .
thingArn	Der Amazon-Ressourcenname (ARN) des AWS IoT Dings, das den AWS IoT Greengrass Kern darstellt.	Sie finden diesen Wert in der AWS IoT Greengrass Konsole unter der Definition für Ihr AWS IoT Ding.
iotHost	Ihr AWS IoT Endpunkt.	Sie finden diesen Wert in der AWS IoT Konsole unter Einstellungen.
ggHost	Ihr AWS IoT Greengrass Endpunkt.	Sie finden diesen Wert in der AWS IoT Konsole unter Einstellungen mit vorangestelltem greengrass . Präfix.
keepAlive	Der MQTT-Zeitraum KeepAlive in Sekunden.	Dies ist ein optionaler Wert. Der Standardwert ist 600 Sekunden.

Feld	Description	Hinweise
useSystemd	Ein Binär-Marker, wenn Ihr Gerät systemd verwendet.	Die Werte sind yes oder no. Verwenden Sie das Abhängigkeitskript in Modul 1 , um festzustellen, ob Ihr Gerät systemd verwendet.

Dienstendpunkte müssen dem Zertifikatstyp der Stammzertifizierungsstelle entsprechen

Ihr AWS IoT Core und Ihre AWS IoT Greengrass Endpunkte müssen dem Zertifikatstyp des Root-CA-Zertifikats auf Ihrem Gerät entsprechen. Wenn die Endpunkte und der Zertifikatstyp nicht übereinstimmen, schlagen Authentifizierungsversuche zwischen dem Gerät und AWS IoT Core oder fehl. AWS IoT Greengrass Weitere Informationen finden Sie unter [Serverauthentifizierung](#) im AWS IoT -Entwicklerhandbuch.

Wenn Ihr Gerät ein Root-CA-Zertifikat von Amazon Trust Services (ATS) verwendet, was die bevorzugte Methode ist, muss es auch ATS-Endpunkte für die Geräteverwaltung und den Betrieb der Discovery-Datenebene verwenden. ATS-Endpunkte enthalten das `ats` Segment, wie in der folgenden Syntax für den AWS IoT Core Endpunkt dargestellt.

```
prefix-ats.iot.region.amazonaws.com
```

Note

Aus Gründen der Abwärtskompatibilität werden AWS IoT Greengrass derzeit in einigen Fällen ältere VeriSign Root-CA-Zertifikate und Endpunkte unterstützt. AWS-Region Wenn Sie ein veraltetes VeriSign Root-CA-Zertifikat verwenden, empfehlen wir Ihnen, einen ATS-Endpunkt zu erstellen und stattdessen ein ATS-Root-CA-Zertifikat zu verwenden. Andernfalls stellen Sie sicher, dass Sie die entsprechenden Legacy-Endpunkte verwenden. Weitere Informationen finden Sie unter [Unterstützte Legacy-Endpunkte](#) im Allgemeine Amazon Web Services-Referenz.

Endpunkte in config.json

Auf einem Greengrass Core-Gerät sind Endpunkte im Objekt `coreThing` in der Datei [config.json](#) angegeben. Die `iotHost` Eigenschaft stellt den AWS IoT Core Endpunkt dar. Die `ggHost` Eigenschaft stellt den AWS IoT Greengrass Endpunkt dar. Im folgenden Beispiel für einen Ausschnitt legen diese Eigenschaften ATS-Endpunkte fest.

```
{
  "coreThing" : {
    ...
    "iotHost" : "abcde1234uvwxyz-ats.iot.us-west-2.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
    ...
  },
}
```

AWS IoT Core Endpunkt

Sie können Ihren AWS IoT Core Endpunkt abrufen, indem Sie den [aws iot describe-endpoint](#) CLI-Befehl mit dem entsprechenden `--endpoint-type` Parameter ausführen.

- Um einen ATS-signierten Endpunkt zurückzugeben, führen Sie Folgendes aus.

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

- Führen Sie Folgendes aus, um einen VeriSign signierten Legacy-Endpunkt zurückzugeben:

```
aws iot describe-endpoint --endpoint-type iot:Data
```

AWS IoT Greengrass Endpunkt

Ihr AWS IoT Greengrass Endpunkt ist Ihr `iotHost` Endpunkt, wobei das Host-Präfix durch Greengrass ersetzt wird. Zum Beispiel lautet der signierte ATS-Endpunkt `greengrass-ats.iot.region.amazonaws.com`. Dies verwendet dieselbe Region wie Ihr AWS IoT Core Endpunkt.

Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Greengrass-Kerne kommunizieren AWS IoT Core mit dem MQTT-Messaging-Protokoll mit TLS-Client-Authentifizierung. Standardmäßig verwendet MQTT über TLS den Port 8883. Doch als

Sicherheitsmaßnahmen können restriktive Umgebungen den ein- und ausgehenden Datenverkehr auf einen kleinen Bereich von TCP-Ports einschränken. Zum Beispiel könnte eine Unternehmens-Firewall Port 443 für HTTPS-Datenverkehr öffnen, andere Ports, die für weniger geläufige Protokolle genutzt werden, wie z. B. Port 8883, für MQTT-Datenverkehr schließen. Bei anderen restriktiven Umgebungen muss der gesamte Datenverkehr möglicherweise über einen HTTP-Proxy laufen, der mit dem Internet verbunden ist.

Um die Kommunikation in diesen Szenarien zu AWS IoT Greengrass ermöglichen, sind die folgenden Konfigurationen möglich:

- MQTT mit TLS-Client-Authentifizierung über Port 443. Wenn Ihr Netzwerk Verbindungen über Port 443 zulässt, können Sie den Core so konfigurieren, dass er anstelle des Standard-Ports 8883 Port 443 für MQTT-Datenverkehr verwendet. Dabei kann es sich um eine direkte Verbindung mit Port 443 oder eine Verbindung über einen Netzwerk-Proxy-Server handeln.

AWS IoT Greengrass verwendet die TLS-Erweiterung [Application Layer Protocol Network](#) (ALPN), um diese Verbindung zu aktivieren. Wie bei der Standardkonfiguration verwendet MQTT über TLS auf Port 443 eine zertifikatbasierte Client-Authentifizierung.

Wenn der Core für die Verwendung einer direkten Verbindung zu Port 443 konfiguriert ist, unterstützt er [AWS IoT Greengrass Softwareupdates over-the-air \(OTA\)](#). Für diese Unterstützung ist AWS IoT Greengrass Core v1.9.3 oder höher erforderlich.

- HTTPS-Kommunikation über Port 443. AWS IoT Greengrass sendet standardmäßig HTTPS-Verkehr über Port 8443, aber Sie können ihn so konfigurieren, dass Port 443 verwendet wird.
- Verbindung über einen Netzwerk-Proxy. Sie können einen Netzwerk-Proxy-Server so konfigurieren, dass er als Vermittler beim Herstellen der Verbindung mit dem Greengrass-Core fungiert. Nur grundlegende Authentifizierung und HTTP- und HTTPS-Proxys werden unterstützt.

Die Proxykonfiguration wird über die `no_proxy` Umgebungsvariablen `http_proxy`, `https_proxy` und an benutzerdefinierte Lambda-Funktionen übergeben. Benutzerdefinierte Lambda-Funktionen müssen diese übergebenen Einstellungen verwenden, um eine Verbindung über den Proxy herzustellen. Gängige Bibliotheken, die von Lambda-Funktionen zum Herstellen von Verbindungen verwendet werden (wie Boto3- oder cURL- und `requests` Python-Pakete), verwenden diese Umgebungsvariablen normalerweise standardmäßig. Wenn eine Lambda-Funktion auch dieselben Umgebungsvariablen spezifiziert, überschreibt sie sie AWS IoT Greengrass nicht.

⚠ Important

Greengrass-Cores, die für die Verwendung eines Netzwerk-Proxys konfiguriert wurden, unterstützen keine [OTA-Aktualisierungen](#).

So konfigurieren Sie MQTT über Port 443

Für diese Funktion ist AWS IoT Greengrass Core v1.7 oder höher erforderlich.

Dieses Verfahren ermöglicht es dem Greengrass-Core, Port 443 für MQTT-Messaging mit AWS IoT Core zu verwenden.

1. Führen Sie den folgenden Befehl aus, um den Greengrass-Daemon zu beenden:

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

2. Öffnen Sie `greengrass-root/config/config.json` zur Bearbeitung als su-Benutzer.
3. Fügen Sie im Objekt `coreThing` die Eigenschaft `iotMqttPort` hinzu und setzen Sie den Wert auf **443**, wie im folgenden Beispiel gezeigt.

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "iotMqttPort" : 443,  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "keepAlive" : 600  
  },  
  ...  
}
```

4. Starten Sie den -Daemon.

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

So konfigurieren Sie HTTPS über Port 443

Für diese Funktion ist AWS IoT Greengrass Core v1.8 oder höher erforderlich.

Dieses Verfahren konfiguriert den Kern so, dass er Port 443 für die HTTPS-Kommunikation verwendet.

1. Führen Sie den folgenden Befehl aus, um den Greengrass-Daemon zu beenden:

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

2. Öffnen Sie **greengrass-root**/config/config.json zur Bearbeitung als su-Benutzer.
3. Fügen Sie im Objekt coreThing die Eigenschaften iotHttpPort und ggHttpPort hinzu, wie im folgenden Beispiel dargestellt.

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "iotHttpPort" : 443,  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "ggHttpPort" : 443,  
    "keepAlive" : 600  
  },  
  ...  
}
```

4. Starten Sie den -Daemon.

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

So konfigurieren Sie einen Netzwerk-Proxy

Für diese Funktion ist AWS IoT Greengrass Core v1.7 oder höher erforderlich.

Dieses Verfahren ermöglicht die AWS IoT Greengrass Verbindung mit dem Internet über einen HTTP- oder HTTPS-Netzwerk-Proxy.

1. Führen Sie den folgenden Befehl aus, um den Greengrass-Daemon zu beenden:

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

2. Öffnen Sie `greengrass-root/config/config.json` zur Bearbeitung als su-Benutzer.
3. Fügen Sie im Objekt `coreThing` das Objekt [networkProxy](#) hinzu, wie im folgenden Beispiel gezeigt.

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "keepAlive" : 600,  
    "networkProxy": {  
      "noProxyAddresses" : "http://128.12.34.56,www.mywebsite.com",  
      "proxy" : {  
        "url" : "https://my-proxy-server:1100",  
        "username" : "Mary_Major",  
        "password" : "pass@word1357"  
      }  
    }  
  },  
  ...  
}
```

4. Starten Sie den -Daemon.

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

networkProxy-Objekt

Geben Sie mithilfe des networkProxy-Objekts Informationen zum Netzwerk-Proxy an. Dieses Objekt hat die folgenden Eigenschaften.

Feld	Description
noProxyAddresses	Optional. Eine durch Komma getrennte Liste von IP-Adressen oder Host-Namen, die vom Proxy ausgeschlossen sind.
proxy	Der Proxy, mit dem die Verbindung hergestellt werden soll. Ein Proxy hat die folgenden Eigenschaften. • <code>url</code>. Die URL des Proxyservers im Formatschema: <code>//userinfo@host:port</code>. • <code>scheme</code>. Das Schema. Es muss sich entweder um <code>http</code> oder <code>https</code> handeln. • <code>userinfo</code>. Fakultativ. Der Benutzername und die Passwortinformationen. Die Felder <code>username</code> und <code>password</code> werden ignoriert, sofern angegeben. • <code>host</code>. Der Hostname oder die IP-Adresse des Proxyservers. • <code>port</code>. Fakultativ. Die Port-Nummer. Wenn nicht angegeben, werden die folgenden Standardwerte verwendet: • <code>http</code>: 80 • <code>https</code>: 443 • <code>username</code>. Fakultativ. Der Benutzername, mit dem die Authentifizierung gegenüber dem Proxy-Server erfolgt.

Feld	Description
	password. Fakultativ. Das Passwort, das zur Authentifizierung am Proxy-Server verwendet wird.

Zulassen von Endpunkten

Die Kommunikation zwischen Greengrass-Geräten AWS IoT Core und/oder AWS IoT Greengrass muss authentifiziert werden. Diese Authentifizierung basiert auf registrierten X.509-Gerätezertifikaten und kryptografischen Schlüsseln. Damit authentifizierte Anfragen ohne zusätzliche Verschlüsselung Proxys durchlaufen können, müssen Sie die folgenden Endpunkte zulassen.

Endpoint	Port	Description
greengrass. <i>region</i> .amazonaws.com	443	Für Operationen auf Steuerebene für die Gruppenverwaltung
<i>prefix</i> -ats.iot. <i>region</i> .amazonaws.com oder <i>prefix</i> .iot. <i>region</i> .amazonaws.com	MQTT: 8883 oder 443 HTTPS: 8443 oder 443	Für Operationen auf Datenebene für das Gerätemanagement, wie z. B. Schattensynchronisierung Erlauben Sie die Verwendung eines

Endpoint	Port	Description
		oder beider Endpunkte, je nachdem, ob Ihre Core- und Client-Geräte (bevorzugte) Root-CA-Zertifikate von Amazon Trust Services, ältere Root-CA-Zertifikate oder beides verwenden. Weitere Informationen finden Sie unter the section called “Dienstendpunkte müssen dem Zertifikatstyp entsprechen” .

Endpoint	Port	Description
greengrass-ats.iot . <i>region</i> .amazonaws.com oder greengrass.iot. <i>region</i> .amazonaws.com	8443 oder 443	Für Geräteerkennungsoptionen Erlauben Sie die Verwendung eines oder beider Endpunkte, je nachdem, ob Ihre Core- und Client-Geräte (bevorzugte) Root-CA-Zertifikate von Amazon Trust Services, ältere Root-CA-Zertifikate oder beides verwenden. Weitere Informationen finden Sie unter the section called “Dienstendpunkte müssen dem Zertifikatstyp entsprechen” .

Endpoint	Port	Description
		 Note Clients, die eine Verbindung über Port 443 herstellen, müssen die TLS-Erweiterung Application Layer Protocol Negotiation (ALPN) implementieren und x-amzn-http-ca als <code>ProtocolName</code> angeben.

Endpoint	Port	Description
		in die übergeben . ProtocolN ameList Weitere Informati onen finden Sie unter Protokoll e im AWS IoT Entwickle rhandbuch .

Endpoint	Port	Description
<code>*.s3.amazonaws.com</code>	443	Wird für Bereitstellungs Vorgänge und over-the-air Updates verwendet. Dieses Format enthält das Zeichen *, da Endpunkte räumlich intern kontrolliert werden und sich jederzeit ändern können.
<code>logs.<i>region</i>.amazonaws.com</code>	443	Erforderlich, wenn die Greengrass-Gruppe so konfiguriert ist, dass Protokolle in CloudWatch geschrieben werden.

Konfigurieren Sie ein Schreibverzeichnis für AWS IoT Greengrass

Diese Funktion ist für AWS IoT Greengrass Core v1.6 und höher verfügbar.

Standardmäßig wird die AWS IoT Greengrass Core-Software in einem einzigen Stammverzeichnis bereitgestellt, in dem alle Lese- und Schreibvorgänge AWS IoT Greengrass ausgeführt werden. Sie können jedoch so konfigurieren, AWS IoT Greengrass dass für alle Schreibvorgänge, einschließlich der Erstellung von Verzeichnissen und Dateien, ein separates Verzeichnis verwendet wird. In diesem Fall werden zwei Verzeichnisse der obersten Ebene AWS IoT Greengrass verwendet:

- Das **greengrass-root**-Verzeichnis, das Sie als Lese-/Schreibzugriff belassen oder optional schreibgeschützt machen können. Dies enthält die AWS IoT Greengrass Core-Software und andere wichtige Komponenten, die während der Laufzeit unveränderlich bleiben sollten, wie Zertifikate und `config.json`
- Das angegebene Schreibverzeichnis. Dies enthält beschreibbaren Inhalt wie Protokolle, Statusinformationen und bereitgestellte benutzerdefinierte Lambda-Funktionen.

Diese Konfiguration führt zu der folgenden Verzeichnisstruktur.

Greengrass-Stammverzeichnis

```
greengrass-root/  
|-- certs/  
|   |-- root.ca.pem  
|   |-- hash.cert.pem  
|   |-- hash.private.key  
|   |-- hash.public.key  
|-- config/  
|   |-- config.json  
|-- ggc/  
|   |-- packages/  
|       |-- package-version/  
|           |-- bin/  
|               |-- daemon  
|               |-- greengrassd  
|               |-- lambda/  
|               |-- LICENSE/  
|               |-- release_notes_package-version.html  
|               |-- runtime/  
|                   |-- java8/  
|                   |-- nodejs8.10/  
|                   |-- python3.8/  
|-- core/
```

Schreibverzeichnis

```
write-directory/
|-- packages/
|   |-- package-version/
|       |-- ggc_root/
|       |-- rootfs_nosys/
|       |-- rootfs_sys/
|       |-- var/
|-- deployment/
|   |-- group/
|       |-- group.json
|   |-- lambda/
|   |-- mlmodel/
|-- var/
|   |-- log/
|   |-- state/
```

So konfigurieren Sie ein Schreibverzeichnis

1. Führen Sie den folgenden Befehl aus, um den Daemon zu beenden: AWS IoT Greengrass

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

2. Öffnen Sie *greengrass-root*/config/config.json zur Bearbeitung als su-Benutzer.
3. Fügen Sie writeDirectory als einen Parameter hinzu und geben Sie, wie im folgenden Beispiel, den Pfad zum Zielverzeichnis an.

```
{
  "coreThing": {
    "caPath": "root-CA.pem",
    "certPath": "hash.pem.crt",
    ...
  },
  ...
  "writeDirectory" : "/write-directory"
}
```

 Note

Sie können die `writeDirectory`-Einstellung so oft aktualisieren, wie Sie möchten. AWS IoT Greengrass verwendet nach der Aktualisierung der Einstellung beim nächsten Start das neu angegebene Schreibverzeichnis, migriert aber keine Inhalte aus dem vorherigen Schreibverzeichnis.

4. Nachdem Ihr Schreibverzeichnis konfiguriert ist, können Sie optional das **greengrass-root**-Verzeichnis auf schreibgeschützt einstellen. Anweisungen finden Sie unter [So wird das Greengrass-Stammverzeichnis auf schreibgeschützt eingestellt](#).

Andernfalls starten Sie den AWS IoT Greengrass Daemon:

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

So aktivieren Sie den Schreibschutz für das Greengrass-Stammverzeichnis

Führen Sie diese Schritte nur aus, wenn das Greengrass-Stammverzeichnis schreibgeschützt sein soll. Das Schreibverzeichnis muss vor Beginn konfiguriert werden.

1. Erteilen Sie Zugriffsberechtigungen für erforderliche Verzeichnisse:
 - a. Dem `config.json`-Besitzer Lese- und Schreibberechtigung erteilen.

```
sudo chmod 0600 /greengrass-root/config/config.json
```

- b. Machen Sie `ggc_user` zum Besitzer der Zertifikate und System-Lambda-Verzeichnisse.

```
sudo chown -R ggc_user:ggc_group /greengrass-root/certs/  
sudo chown -R ggc_user:ggc_group /greengrass-root/ggc/packages/1.11.6/lambda/
```

 Note

Die Konten `ggc_user` und `ggc_group` werden standardmäßig verwendet, um Lambda-Systemfunktionen auszuführen. Wenn Sie die [Standardzugriffsidentität](#) auf

Gruppenebene konfiguriert haben, um verschiedene Konten zu verwenden, sollten Sie stattdessen dem Benutzer (UID) und der Gruppe (GID) Berechtigungen erteilen.

2. Machen Sie das *greengrass-root*-Verzeichnis schreibgeschützt, indem Sie Ihren bevorzugten Mechanismus verwenden.

 Note

Sie können den Schreibschutz für das *greengrass-root*-Verzeichnis einstellen, indem Sie das Verzeichnis als schreibgeschützt mounten. Um jedoch over-the-air (OTA) -Updates auf die AWS IoT Greengrass Core-Software in einem bereitgestellten Verzeichnis anzuwenden, muss das Verzeichnis zuerst entfernt und nach dem Update erneut bereitgestellt werden. Sie können diese `umount`- und `mount`-Operationen für die `ota_pre_update`- und `ota_post_update`-Skripts hinzufügen. Weitere Informationen über OTA-Updates finden Sie unter [the section called “Greengrass OTA-Update-Agent”](#) und [the section called “Managed Respawn mit OTA-Updates”](#).

3. Starten Sie den -Daemon.

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

Wenn die Berechtigungen aus Schritt 1 nicht korrekt festgelegt wurden, wird der Daemon nicht gestartet.

Konfigurieren der MQTT-Einstellungen

In der AWS IoT Greengrass Umgebung können lokale Client-Geräte, Lambda-Funktionen, Konnektoren und Systemkomponenten miteinander und mit AWS IoT Core kommunizieren. Die gesamte Kommunikation erfolgt durch den Core, der die [Abonnements](#) verwaltet, die die MQTT-Kommunikation zwischen Entitäten autorisieren.

Informationen zu MQTT-Einstellungen AWS IoT Greengrass, für die Sie konfigurieren können, finden Sie in den folgenden Abschnitten:

- [the section called “Nachrichtenqualität des Service”](#)
- [the section called “MQTT-Nachrichtenwarteschlange”](#)

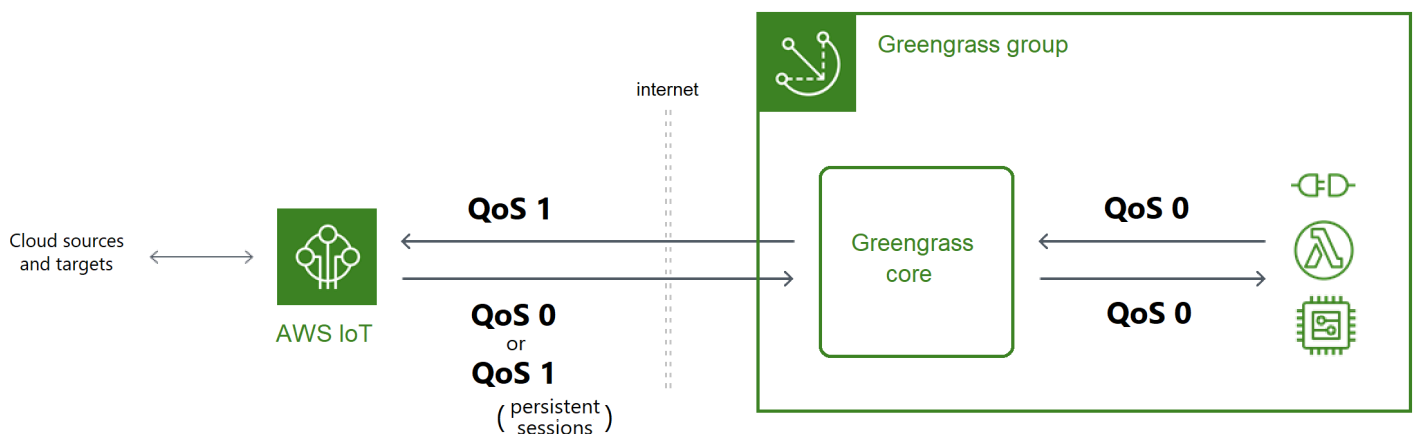
- [the section called “Persistente MQTT-Sitzungen mit AWS IoT Core”](#)
- [the section called “Client IDs für MQTT-Verbindungen mit AWS IoT”](#)
- [MQTT-Port für lokales Messaging](#)
- [the section called “Timeout für Veröffentlichungs-, Abonnierungs- und Abmeldevorgänge in MQTT-Verbindungen mit dem AWS Cloud”](#)

Note

OPC-UA ist ein Standard für den Informationsaustausch für die industrielle Kommunikation. [Um die Unterstützung für OPC-UA auf dem Greengrass-Core zu implementieren, können Sie den IoT-Connector verwenden. SiteWise](#) Der Konnektor sendet Daten von Industriegeräten von OPC-UA-Servern an Anlagen in. AWS IoT SiteWise

Nachrichtenqualität des Service

AWS IoT Greengrass unterstützt die Quality-of-Service-Stufen (QoS) 0 oder 1, abhängig von Ihrer Konfiguration und dem Ziel und der Richtung der Kommunikation. Der Greengrass-Kern fungiert als Client für die Kommunikation mit dem lokalen Netzwerk AWS IoT Core und als Nachrichtenbroker für die Kommunikation im lokalen Netzwerk.



Weitere Informationen zu MQTT und QoS finden Sie unter [Getting Started](#) auf der MQTT-Website.

Kommunikation mit dem AWS Cloud

- Ausgehende Nachrichten verwenden QoS 1

Der Kern sendet Nachrichten, die für AWS Cloud Ziele bestimmt sind, die QoS 1 verwenden. AWS IoT Greengrass verwendet eine MQTT-Nachrichtenwarteschlange, um diese Nachrichten zu verarbeiten. Wenn die Nachrichtenzustellung von nicht bestätigt wird AWS IoT, wird die Nachricht in einem Spoolmodus gespeichert, sodass sie zu einem späteren Zeitpunkt erneut versucht wird. Die Nachricht kann nicht erneut versucht werden, wenn die Warteschlange voll ist. Die Bestätigung der Nachrichtenzustellung kann dazu beitragen, den Datenverlust aufgrund unterbrochener Konnektivität zu minimieren.

Da ausgehende Nachrichten QoS 1 AWS IoT verwenden sollen, hängt die maximale Geschwindigkeit, mit der der Greengrass-Core Nachrichten senden kann, von der Latenz zwischen dem Kern und ab. AWS IoT Jedes Mal, wenn der Core eine Nachricht sendet, wartet er, bis er die Nachricht AWS IoT bestätigt, bevor er die nächste Nachricht sendet. Wenn die Umlaufzeit zwischen dem Kern und seinem Kern beispielsweise 50 Millisekunden AWS-Region beträgt, kann der Kern bis zu 20 Nachrichten pro Sekunde senden. Berücksichtigen Sie dieses Verhalten, wenn Sie auswählen, AWS-Region wo Ihr Kern eine Verbindung herstellt. Um umfangreiche IoT-Daten in den aufzunehmen AWS Cloud, können Sie den [Stream](#) Manager verwenden.

Weitere Informationen zur MQTT-Nachrichtenwarteschlange, einschließlich der Konfiguration eines lokalen Speichercaches, der Nachrichten, die für Ziele bestimmt sind, speichern kann, finden Sie unter. AWS Cloud [the section called “MQTT-Nachrichtenwarteschlange”](#)

- Eingehende Nachrichten verwenden QoS 0 (Standard) oder QoS 1

Standardmäßig abonniert der Core mit QoS 0 Nachrichten aus AWS Cloud Quellen. Wenn Sie persistente Sitzungen aktivieren, abonniert der Core mit QoS 1. Dies kann dazu beitragen, Datenverluste durch intermittierende Konnektivität zu minimieren. Um die QoS für diese Abonnements zu verwalten, konfigurieren Sie Persistenzeinstellungen für die lokale Spooler-Systemkomponente.

Weitere Informationen, unter anderem dazu, wie Sie es dem Core ermöglichen, eine persistente Sitzung mit AWS Cloud Zielen einzurichten, finden Sie unter. [the section called “Persistente MQTT-Sitzungen mit AWS IoT Core”](#)

Kommunikation mit lokalen Zielen

Die gesamte lokale Kommunikation verwendet QoS 0. Der Core versucht einmal, eine Nachricht an ein lokales Ziel zu senden, bei dem es sich um eine Greengrass Lambda-Funktion, einen Connector oder ein [Client-Gerät](#) handeln kann. Der Core speichert keine Nachrichten und

bestätigt die Zustellung nicht. Nachrichten können überall zwischen den Komponenten gelöscht werden.

 Note

Obwohl die direkte Kommunikation zwischen Lambda-Funktionen kein MQTT-Messaging verwendet, ist das Verhalten dasselbe.

MQTT-Nachrichtenwarteschlange für Cloud-Ziele

MQTT-Nachrichten, die für AWS Cloud Ziele bestimmt sind, werden in eine Warteschlange gestellt, um auf ihre Verarbeitung zu warten. Nachrichten in der Warteschlange werden in der FIFO (First-in-First-out)-Reihenfolge verarbeitet. Nachdem eine Nachricht verarbeitet und veröffentlicht wurde AWS IoT Core, wird sie aus der Warteschlange entfernt.

Standardmäßig speichert der Greengrass-Kern unverarbeitete Nachrichten, die für Ziele bestimmt sind, im Speicher. AWS Cloud Sie können den Core so konfigurieren, dass nicht verarbeitete Nachrichten stattdessen in einem lokalen Speichercache gespeichert werden. Im Gegensatz zum In-Memory-Speicher kann der lokale Speichercache auch nach einem Neustart des Kerns bestehen bleiben (z. B. nach einer Gruppenbereitstellung oder einem Gerätereustart), sodass die Nachrichten weiter verarbeitet AWS IoT Greengrass werden können. Sie können auch die Speichergröße konfigurieren.

 Warning

Der Greengrass-Core könnte doppelte MQTT-Nachrichten in die Warteschlange stellen, wenn er die Verbindung verliert, weil er einen Veröffentlichungsvorgang wiederholt, bevor der MQTT-Client erkennt, dass er offline ist. Um doppelte MQTT-Nachrichten für Cloud-Ziele zu vermeiden, konfigurieren Sie den `keepAlive` Wert des Kerns auf weniger als die Hälfte seines Werts. `mqttOperationTimeout` Weitere Informationen finden Sie unter [AWS IoT Greengrass Kernkonfigurationsdatei](#).

AWS IoT Greengrass verwendet die Spooler-Systemkomponente (die `GGCloudSpooler` Lambda-Funktion), um die Nachrichtenwarteschlange zu verwalten. Sie können die folgenden `GGCloudSpooler`-Umgebungsvariablen verwenden, um Speichereinstellungen zu konfigurieren.

- `GG_CONFIG_STORAGE_TYPE`. Der Speicherort der Nachrichtenwarteschlange. Die folgenden Werte sind gültig:
 - `FileSystem`. Speichern Sie unverarbeitete Nachrichten im lokalen Speichercache auf der Festplatte des physischen Kerngeräts. Wenn der Core neu gestartet wird, werden Nachrichten in der Warteschlange für die Verarbeitung beibehalten. Nachrichten werden nach der Verarbeitung entfernt.
 - `Memory (default)`. Speichern Sie nicht verarbeitete Nachrichten im Arbeitsspeicher. Wenn der Core neu gestartet wird, gehen Nachrichten in der Warteschlange verloren.

Diese Option sind für Geräte mit eingeschränkten Hardwaremöglichkeiten optimiert. Wenn Sie diese Konfiguration verwenden, sollten Sie Gruppen bereitstellen oder das Gerät neu starten, wenn die Serviceunterbrechung möglichst gering ist.

- `GG_CONFIG_MAX_SIZE_BYTES`. Die Speichergröße in Bytes. Bei diesem Wert kann es sich um eine beliebige nicht negative Ganzzahl handeln, die größer oder gleich 262144 (256 KB) ist. Eine kleinere Größe verhindert, dass die AWS IoT Greengrass Core-Software gestartet wird. Die Standardgröße ist 2.5 MB. Wenn die maximale Größe erreicht ist, werden die ältesten Nachrichten in der Warteschlange durch neue Nachrichten ersetzt.

Note

Diese Funktion ist für AWS IoT Greengrass Core v1.6 und höher verfügbar. Frühere Versionen verwenden In-Memory-Speicher mit einer Warteschlangengröße von 2,5 MB. Sie können keine Speichereinstellungen für frühere Versionen konfigurieren.

So speichern Sie Nachrichten im lokalen Speicher

Sie können so konfigurieren AWS IoT Greengrass , dass Nachrichten im Dateisystem zwischengespeichert werden, sodass sie auch nach einem Neustart des Kerns bestehen bleiben. Dazu stellen Sie eine Funktionsdefinitionsversion bereit, in der die `GGCloudSpooler`-Funktion den Speichertyp auf `FileSystem` festlegt. Sie müssen die AWS IoT Greengrass API verwenden, um den lokalen Speichercache zu konfigurieren. Sie können dies von der Konsole aus erledigen.

Das folgende Verfahren verwendet den [create-function-definition-version](#) CLI-Befehl, um den Spooler so zu konfigurieren, dass Nachrichten in der Warteschlange im Dateisystem gespeichert werden. Außerdem wird damit eine Warteschlange mit einer Größe von 2,6 MB konfiguriert.

1. Rufen Sie IDs die Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

 Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die LatestVersionWerte und Id aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.
 - Ersetzen Sie *group-id* durch den Id, den Sie kopiert haben.
 - Ersetzen Sie *latest-group-version-id* durch den LatestVersion, den Sie kopiert haben.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. Kopieren Sie aus dem Definition Objekt in der Ausgabe die CoreDefinitionVersionArn und die ARNs aller anderen Gruppenkomponenten außer FunctionDefinitionVersionArn. Diese Werte verwenden Sie, wenn Sie eine neue Gruppenversion erstellen.
5. Kopieren Sie die ID der Funktionsdefinition aus FunctionDefinitionVersionArn. Die ID ist die GUID, die dem functions-Segment im ARN folgt, wie im folgenden Beispiel gezeigt.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

Note

Sie können auch eine Funktionsdefinition erstellen, indem Sie den [create-function-definition](#) Befehl ausführen und dann die ID aus der Ausgabe kopieren.

6. Fügen Sie der Funktionsdefinition eine Funktionsdefinitionsversion hinzu.

- *function-definition-id* Ersetzen Sie sie durch Id die, die Sie für die Funktionsdefinition kopiert haben.
- *arbitrary-function-id* Ersetzen Sie durch einen Namen für die Funktion, z. **spooler-function B**.
- Fügen Sie dem functions Array alle Lambda-Funktionen hinzu, die Sie in diese Version aufnehmen möchten. Sie können den [get-function-definition-version](#) Befehl verwenden, um die Greengrass Lambda-Funktionen aus einer vorhandenen Funktionsdefinitionsversion abzurufen.

Warning

Stellen Sie sicher, dass Sie einen Wert für GG_CONFIG_MAX_SIZE_BYTES angeben, der größer als oder gleich 262144 ist. Eine geringere Größe verhindert, dass die AWS IoT Greengrass Core-Software gestartet wird.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
"arn:aws:lambda:::function:GGCloudSpooler:1", "FunctionConfiguration":
{"Environment": {"Variables":
{"GG_CONFIG_MAX_SIZE_BYTES": "2621440", "GG_CONFIG_STORAGE_TYPE": "FileSystem"}}, "Executable":
"spooler", "MemorySize": 32768, "Pinned": true, "Timeout": 3}, {"Id": "arbitrary-
function-id"}]'
```

 Note

Wenn Sie die `GG_CONFIG_SUBSCRIPTION_QUALITY` Umgebungsvariable zuvor so eingestellt haben, dass sie [persistente Sitzungen unterstützt AWS IoT Core](#), nehmen Sie sie in diese Funktionsinstanz auf.

7. Kopieren Sie den Arn der Funktionsdefinitionsversion aus der Ausgabe.
8. Erstellen Sie eine Gruppenversion, die die Lambda-Funktion des Systems enthält.
 - Ersetzen Sie *group-id* durch die Id für die Gruppe.
 - *core-definition-version-arn* Ersetzen Sie es durch `CoreDefinitionVersionArn` das, was Sie aus der neuesten Gruppenversion kopiert haben.
 - *function-definition-version-arn* Ersetzen Sie es durch Arn das, was Sie für die neue Version der Funktionsdefinition kopiert haben.
 - Ersetzen Sie die ARNs für andere Gruppenkomponenten (z. B. `SubscriptionDefinitionVersionArn` oder `DeviceDefinitionVersionArn`), die Sie aus der neuesten Gruppenversion kopiert haben.
 - Entfernen Sie alle nicht verwendeten Parameter. Entfernen Sie zum Beispiel `--resource-definition-version-arn`, wenn Ihre Gruppenversion keine Ressourcen enthält.

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Kopieren Sie die Version aus der Ausgabe. Dies ist die ID der neuen Gruppenversion.
10. Stellen Sie die Gruppe mit der neuen Gruppenversion bereit.
 - *group-id* Ersetzen Sie sie durch Id die, die Sie für die Gruppe kopiert haben.
 - *group-version-id* Ersetzen Sie es durch Version das, was Sie für die neue Gruppenversion kopiert haben.

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

Um die Speichereinstellungen zu aktualisieren, verwenden Sie die AWS IoT Greengrass API, um eine neue Version der Funktionsdefinition zu erstellen, die die GGCloudSpooler Funktion mit der aktualisierten Konfiguration enthält. Fügen Sie anschließend die Funktionsdefinitionsversion einer neuen Gruppenversion (zusammen mit Ihren anderen Gruppenkomponenten) hinzu und stellen Sie die Gruppenversion bereit. Wenn Sie die Standardkonfiguration wiederherstellen möchten, können Sie eine Funktionsdefinitionsversion bereitstellen, in der die GGCloudSpooler-Funktion nicht enthalten ist.

Diese System-Lambda-Funktion ist in der Konsole nicht sichtbar. Nachdem die Funktion der neuesten Gruppenversion hinzugefügt wurde, ist sie allerdings in Bereitstellungen enthalten, die Sie von der Konsole aus erstellen, es sei denn, Sie ersetzen oder verschieben sie mit der API.

Persistente MQTT-Sitzungen mit AWS IoT Core

Diese Funktion ist für AWS IoT Greengrass Core v1.10 und höher verfügbar.

Ein Greengrass-Core kann eine persistente Sitzung mit dem AWS IoT Message Broker aufbauen. Eine dauerhafte Sitzung ist eine fortlaufende Verbindung, die es dem Core ermöglicht, Nachrichten zu empfangen, die gesendet werden, während der Core offline ist. Der Core ist der Client in der Verbindung.

In einer persistenten Sitzung speichert der AWS IoT Message Broker alle Abonnements, die der Core während der Verbindung abschließt. [Wenn der Core die Verbindung trennt, speichert der AWS IoT Message Broker unbestätigte und neue Nachrichten, die als QoS 1 veröffentlicht wurden und für lokale Ziele wie Lambda-Funktionen und Client-Geräte bestimmt sind.](#) Wenn der Core wieder eine Verbindung herstellt, wird die persistente Sitzung wieder aufgenommen und der AWS IoT Message Broker sendet gespeicherte Nachrichten mit einer maximalen Rate von 10 Nachrichten pro Sekunde an den Core. Dauerhafte Sitzungen haben eine Standardablauffrist von 1 Stunde, die beginnt, wenn der Nachrichtenbroker erkennt, dass der Core getrennt wird. Weitere Informationen finden Sie unter [Persistente MQTT-Sitzungen im AWS IoT Entwicklerhandbuch.](#)

AWS IoT Greengrass verwendet die Spooler-Systemkomponente (die GGCloudSpooler Lambda-Funktion), um Abonnements zu erstellen, die AWS IoT als Quelle haben. Sie können die folgende GGCloudSpooler-Umgebungsvariable verwenden, um persistente Sitzungen zu konfigurieren.

- `GG_CONFIG_SUBSCRIPTION_QUALITY`. Die Qualität von Abonnements, die AWS IoT als Quelle verwenden. Die folgenden Werte sind gültig:
 - `AtMostOnce` (default). Deaktiviert persistente Sitzungen. Abonnements verwenden QoS 0.
 - `AtLeastOncePersistent`. Aktiviert persistente Sitzungen. Legt das `cleanSession`-Flag in CONNECT-Nachrichten auf 0 fest und abonniert mit QoS 1.

Nachrichten, die mit QoS 1 veröffentlicht wurden, die der Core erhält, erreichen garantiert die In-Memory-Arbeitswarteschlange des Greengrass-Daemons. Der Core bestätigt die Nachricht, nachdem sie zur Warteschlange hinzugefügt wurde. Die nachfolgende Kommunikation von der Warteschlange zum lokalen Ziel (z. B. Greengrass Lambda-Funktion, Connector oder Gerät) wird als QoS 0 gesendet. AWS IoT Greengrass garantiert nicht die Zustellung an lokale Ziele.

Note

Sie können die Konfigurationseigenschaft [maxWorkItemCount](#) verwenden, um die Größe der Warteschlange für Arbeitsaufgaben zu steuern. Beispielsweise können Sie die Warteschlangengröße erhöhen, wenn Ihre Arbeitslast einen hohen MQTT-Datenverkehr erfordert.

Wenn persistente Sitzungen aktiviert sind, öffnet der Core mindestens eine zusätzliche Verbindung für den MQTT-Nachrichtenaustausch mit AWS IoT. Weitere Informationen finden Sie unter [the section called “Client IDs für MQTT-Verbindungen mit AWS IoT”](#).

So konfigurieren Sie persistente MQTT-Sitzungen

Sie können konfigurieren AWS IoT Greengrass, dass persistente Sitzungen mit AWS IoT Core verwendet werden. Dazu stellen Sie eine Funktionsdefinitionsversion bereit, in der die GGCloudSpooler-Funktion die Abonnementqualität auf `AtLeastOncePersistent` festlegt. Diese Einstellung gilt für alle Ihre Abonnements, die AWS IoT Core (`cloud`) als Quelle haben. Sie müssen die AWS IoT Greengrass API verwenden, um persistente Sitzungen zu konfigurieren. Sie können dies von der Konsole aus erledigen.

Das folgende Verfahren verwendet den [create-function-definition-version](#) CLI-Befehl, um den Spooler für die Verwendung persistenter Sitzungen zu konfigurieren. In diesem Verfahren wird davon ausgegangen, dass Sie die Konfiguration auf die neueste Gruppenversion einer vorhandenen Gruppe aktualisieren.

1. Rufen Sie IDs der Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die LatestVersionWerte und Id aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.
 - Ersetzen Sie *group-id* durch den Id, den Sie kopiert haben.
 - Ersetzen Sie *latest-group-version-id* durch den LatestVersion, den Sie kopiert haben.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. Kopieren Sie aus dem Definition Objekt in der Ausgabe die CoreDefinitionVersionArn und die ARNs aller anderen Gruppenkomponenten außer FunctionDefinitionVersionArn. Diese Werte verwenden Sie, wenn Sie eine neue Gruppenversion erstellen.

5. Kopieren Sie die ID der Funktionsdefinition aus `FunctionDefinitionVersionArn`. Die ID ist die GUID, die dem `functions`-Segment im ARN folgt, wie im folgenden Beispiel gezeigt.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

Note

Sie können auch eine Funktionsdefinition erstellen, indem Sie den [create-function-definition](#)Befehl ausführen und dann die ID aus der Ausgabe kopieren.

6. Fügen Sie der Funktionsdefinition eine Funktionsdefinitionsversion hinzu.
 - *function-definition-id* Ersetzen Sie sie durch Id die, die Sie für die Funktionsdefinition kopiert haben.
 - *arbitrary-function-id* Ersetzen Sie durch einen Namen für die Funktion, z. **spooler-function B**.
 - Fügen Sie dem `functions` Array alle Lambda-Funktionen hinzu, die Sie in diese Version aufnehmen möchten. Sie können den [get-function-definition-version](#)Befehl verwenden, um die Greengrass Lambda-Funktionen aus einer vorhandenen Funktionsdefinitionsversion abzurufen.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
"arn:aws:lambda::function:GGCloudSpooler:1", "FunctionConfiguration":
{"Environment": {"Variables":
{"GG_CONFIG_SUBSCRIPTION_QUALITY": "AtLeastOncePersistent"}}, "Executable":
"spooler", "MemorySize": 32768, "Pinned": true, "Timeout": 3}, "Id": "arbitrary-
function-id"}]'
```

Note

Wenn Sie zuvor die Umgebungsvariablen `GG_CONFIG_STORAGE_TYPE` oder `GG_CONFIG_MAX_SIZE_BYTES` zum [Definieren von Speichereinstellungen](#) festlegen, schließen Sie sie in diese Funktions-Instance ein.

7. Kopieren Sie den Arn der Funktionsdefinitionsversion aus der Ausgabe.
8. Erstellen Sie eine Gruppenversion, die die Lambda-Funktion des Systems enthält.
 - Ersetzen Sie *group-id* durch die Id für die Gruppe.
 - *core-definition-version-arn* Ersetzen Sie es durch CoreDefinitionVersionArn das, was Sie aus der neuesten Gruppenversion kopiert haben.
 - *function-definition-version-arn* Ersetzen Sie es durch Arn das, was Sie für die neue Version der Funktionsdefinition kopiert haben.
 - Ersetzen Sie die ARNs für andere Gruppenkomponenten (z. B. SubscriptionDefinitionVersionArn oder DeviceDefinitionVersionArn), die Sie aus der neuesten Gruppenversion kopiert haben.
 - Entfernen Sie alle nicht verwendeten Parameter. Entfernen Sie zum Beispiel --resource-definition-version-arn, wenn Ihre Gruppenversion keine Ressourcen enthält.

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Kopieren Sie die Version aus der Ausgabe. Dies ist die ID der neuen Gruppenversion.
10. Stellen Sie die Gruppe mit der neuen Gruppenversion bereit.
 - *group-id* Ersetzen Sie sie durch Id die, die Sie für die Gruppe kopiert haben.
 - *group-version-id* Ersetzen Sie es durch Version das, was Sie für die neue Gruppenversion kopiert haben.

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

11. (Optional) Erhöhen Sie die Eigenschaft [maxWorkItemCount](#) in der Core-Konfigurationsdatei. Dies kann dem Core helfen, erhöhten MQTT-Datenverkehr und die Kommunikation mit lokalen Zielen zu bewältigen.

Um den Core mit diesen Konfigurationsänderungen zu aktualisieren, verwenden Sie die AWS IoT Greengrass API, um eine neue Version der Funktionsdefinition zu erstellen, die die `GGCloudSpooler` Funktion mit der aktualisierten Konfiguration enthält. Fügen Sie anschließend die Funktionsdefinitionsversion einer neuen Gruppenversion (zusammen mit Ihren anderen Gruppenkomponenten) hinzu und stellen Sie die Gruppenversion bereit. Wenn Sie die Standardkonfiguration wiederherstellen möchten, können Sie eine Funktionsdefinitionsversion erstellen, in der die `GGCloudSpooler`-Funktion nicht enthalten ist.

Diese System-Lambda-Funktion ist in der Konsole nicht sichtbar. Nachdem die Funktion der neuesten Gruppenversion hinzugefügt wurde, ist sie allerdings in Bereitstellungen enthalten, die Sie von der Konsole aus erstellen, es sei denn, Sie ersetzen oder verschieben sie mit der API.

Client IDs für MQTT-Verbindungen mit AWS IoT

Diese Funktion ist für AWS IoT Greengrass Core v1.8 und höher verfügbar.

Der Greengrass-Kern öffnet MQTT-Verbindungen AWS IoT Core für Operationen wie Shadow Sync und Zertifikatsverwaltung. Für diese Verbindungen generiert der Core einen vorhersehbaren Client, der auf dem IDs Namen des Kern-Dings basiert. Der IDs Predictable Client kann mit Überwachungs-, Prüf- und Preisfunktionen, einschließlich AWS IoT Device Defender [AWS IoT Lebenszyklusereignissen](#), verwendet werden. Sie können auch Logik rund um den Predictable Client erstellen IDs (z. B. Vorlagen für [Abonnementrichtlinien](#), die auf Zertifikatsattributen basieren).

GGC v1.9 and later

Zwei Greengrass-Systemkomponenten öffnen MQTT-Verbindungen mit AWS IoT Core Diese Komponenten verwenden die folgenden Muster, um den Client IDs für die Verbindungen zu generieren.

Operation	Client-ID Muster
Bereitstellungen	<i>core-thing-name</i>
	Beispiel: MyCoreThing

Operation	Client-ID Muster
	Verwenden Sie diese Client-ID für Benachrichtigungen bei folgenden Lebenszykluseignissen: Verbinden, Verbindung trennen, Abonnieren und Abonnement kündigen.
Subscriptions (Abonnements)	<i>core-thing-name</i> -<i>cn</i> Beispiel: MyCoreThing-c01 <i>n</i> ist eine Ganzzahl, die bei 00 beginnt und bei jeder neuen Verbindung auf eine maximale Anzahl von 250 erhöht wird. Die Anzahl der Verbindungen wird durch die Anzahl der Geräte bestimmt, mit denen ihr Schattens tatus synchronisiert wird AWS IoT Core (maximal 2.500 pro Gruppe), und durch die Anzahl der Abonnements, deren Quelle in der Gruppe ist (maximal 10.000 pro Gruppe). cloud Die Spooler-Systemkomponente stellt eine Verbindung her AWS IoT Core , um Nachrichten für Abonnements mit einer Cloud-Quelle oder einem Cloud-Ziel auszutauschen. Der Spooler fungiert auch als Proxy für den Nachrichtenaustausch zwischen dem lokalen Shadow-Dienst AWS IoT Core und dem Gerätezertifikatsmanager.

Verwenden Sie die folgende Formel, um die Anzahl der MQTT-Verbindungen pro Gruppe zu berechnen:

```
number of MQTT connections per group = number of connections for
Deployment Agent + number of connections for Subscriptions
```

Wobei gilt,

- Anzahl der Verbindungen für Deployment Agent = 1.
- Anzahl der Verbindungen für Abonnements = $(2 \text{ subscriptions for supporting certificate generation} + \text{number of MQTT topics in AWS IoT Core} + \text{number of device shadows synced}) / 50$.
- Wobei, 50 = die maximale Anzahl von Abonnements pro Verbindung, die unterstützt AWS IoT Core werden können.

Note

Wenn Sie [persistente Sitzungen](#) für das Abonnement aktivieren AWS IoT Core, öffnet der Core mindestens eine zusätzliche Verbindung, die in einer persistenten Sitzung verwendet werden kann. Die Systemkomponenten unterstützen keine persistenten Sitzungen, sodass sie diese Verbindung nicht gemeinsam nutzen können.

Um die Anzahl der MQTT-Verbindungen zu reduzieren und die Kosten zu senken, können Sie lokale Lambda-Funktionen verwenden, um Daten am Edge zu aggregieren. Dann senden Sie die aggregierten Daten an die AWS Cloud. Dadurch verwenden Sie weniger MQTT-Themen in AWS IoT Core. Weitere Informationen finden Sie unter [AWS IoT Greengrass – Preise](#).

GGC v1.8

Mehrere Greengrass-Systemkomponenten öffnen MQTT-Verbindungen mit AWS IoT Core. Diese Komponenten verwenden die folgenden Muster, um den Client IDs für die Verbindungen zu generieren.

Operation	Client-ID Muster
Bereitstellungen	<i>core-thing-name</i> Beispiel: MyCoreThing Verwenden Sie diese Client-ID für Benachrichtigungen bei folgenden Lebenszykluseignissen: Verbinden, Verbindung trennen, Abonnieren und Abonnement kündigen.

Operation	Client-ID Muster
MQTT-Nachrichtenaustausch mit AWS IoT Core	<i>core-thing-name</i> -spr Beispiel: MyCoreThing-spr
Schattensynchronisierung	<i>core-thing-name</i> -snn Beispiel: MyCoreThing-s01 <i>nn</i> ist eine Ganzzahl, die bei 00 beginnt und mit jeder neuen Verbindung auf höchstens 03 ansteigt. Die Anzahl der Verbindungen wird durch die Anzahl der Geräte (maximal 200 Geräte pro Gruppe) bestimmt, mit denen ihr Schattenstatus synchronisiert wird AWS IoT Core (maximal 50 Abonnements pro Verbindung).
Gerätezertifikatsverwaltung	<i>core-thing-name</i> -dcm Beispiel: MyCoreThing-dcm

Note

Doppelte Clients, die bei gleichzeitigen Verbindungen IDs verwendet werden, können zu einer unendlichen Verbindungs-/Trennschleife führen. Dies kann der Fall sein, wenn für ein anderes Gerät festgelegt ist, dass der Core-Gerätenamen als Client-ID in Verbindungen verwendet wird. Weitere Informationen finden Sie im [Fehlerbehandlungsschritt](#).

Greengrass-Geräte sind zudem vollständig in den Flotten-Indizierungsdienst von AWS IoT Device Management integriert. Auf diese Weise können Sie Geräte in der Cloud auf der Grundlage von Geräteattributen, Schattenzustand und Verbindungsstatus indizieren und suchen. Greengrass-Geräte stellen beispielsweise mindestens eine Verbindung her, bei der der Objektname als Client-ID verwendet wird, damit Sie mithilfe der Gerätekonnektivitätsindizierung ermitteln können, welche Greengrass-Geräte derzeit mit AWS IoT Core verbunden oder davon getrennt sind. Weitere Informationen finden Sie unter [Fleet Indexing Service im Entwicklerhandbuch](#). AWS IoT

Konfigurieren des MQTT-Ports für lokales Messaging

Für diese Funktion ist AWS IoT Greengrass Core v1.10 oder höher erforderlich.

[Der Greengrass-Kern fungiert als lokaler Nachrichtenbroker für MQTT-Messaging zwischen lokalen Lambda-Funktionen, Konnektoren und Client-Geräten.](#) Standardmäßig verwendet der Core Port 8883 für MQTT-Datenverkehr im lokalen Netzwerk. Möglicherweise möchten Sie den Port ändern, um Konflikte mit anderer Software zu vermeiden, die auf Port 8883 ausgeführt wird.

So konfigurieren Sie die Portnummer, die der Core für den lokalen MQTT-Datenverkehr verwendet

1. Führen Sie den folgenden Befehl aus, um den Greengrass-Daemon zu beenden:

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

2. Öffnen Sie **greengrass-root**/config/config.json zur Bearbeitung als su-Benutzer.
3. Fügen Sie im coreThing-Objekt die ggMqttPort-Eigenschaft hinzu und legen Sie den Wert auf die Portnummer fest, die Sie verwenden möchten. Gültige Werte sind 1024 bis 65535. Im folgenden Beispiel wird die Portnummer auf 9000 festgelegt.

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "12345abcde.cert.pem",  
    "keyPath" : "12345abcde.private.key",  
    "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",  
    "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",  
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",  
    "ggMqttPort" : 9000,  
    "keepAlive" : 600  
  },  
  ...  
}
```

4. Starten Sie den -Daemon.

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


5. Wenn die [automatische IP-Erkennung](#) für den Core aktiviert ist, ist die Konfiguration abgeschlossen.

Wenn die automatische IP-Erkennung nicht aktiviert ist, müssen Sie die Konnektivitätsinformationen für den Core aktualisieren. Auf diese Weise können Client-Geräte bei Erkennungsvorgängen die richtige Portnummer erhalten, um wichtige Verbindungsinformationen zu erhalten. Sie können die AWS IoT Konsole oder AWS IoT Greengrass API verwenden, um die Informationen zur Kernkonnektivität zu aktualisieren. Für dieses Verfahren aktualisieren Sie nur die Portnummer. Die lokale IP-Adresse für den Core bleibt unverändert.

So aktualisieren Sie die Konnektivitätsinformationen für den Core (Konsole)

1. Wählen Sie auf der Gruppenkonfigurationsseite den Greengrass-Core aus.
2. Wählen Sie auf der Seite mit den Kerndetails den Tab MQTT-Broker-Endpoints aus.
3. Wählen Sie Endpoints verwalten und dann Endpoint hinzufügen
4. Geben Sie Ihre aktuelle lokale IP-Adresse und die neue Portnummer ein. Im folgenden Beispiel wird die Portnummer 9000 für die IP-Adresse 192.168.1.8 festgelegt.
5. Entfernen Sie den veralteten Endpunkt und wählen Sie dann Aktualisieren aus.

So aktualisieren Sie die Konnektivitätsinformationen für den Core (API)

- Verwenden Sie die Aktion „[UpdateConnectivityInfo](#)“. Im folgenden Beispiel wird `update-connectivity-info` in der AWS CLI verwendet, um die Portnummer 9000 für die IP-Adresse 192.168.1.8 festzulegen.

```
aws greengrass update-connectivity-info \
  --thing-name "MyGroup_Core" \
  --connectivity-info "[{\"Metadata\":\"\",\"PortNumber\":9000,
  \"HostAddress\":\"192.168.1.8\",\"Id\":\"localIP_192.168.1.8\"},{\"Metadata
  \":\",\"PortNumber\":8883,\"HostAddress\":\"127.0.0.1\",\"Id\":
  \"localhost_127.0.0.1_0\"}]"
```

 Note

Sie können auch den Port konfigurieren, den der Core für MQTT-Messaging verwendet. AWS IoT Core Weitere Informationen finden Sie unter [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).

Timeout für Veröffentlichungs-, Abonnierungs- und Abmeldevorgänge in MQTT-Verbindungen mit dem AWS Cloud

Diese Funktion ist in AWS IoT Greengrass Version 1.10.2 oder höher verfügbar.

Sie können die Zeit (in Sekunden) so konfigurieren, dass der Greengrass-Core eine Veröffentlichungs-, Abonnement- oder Abmeldeoperation in MQTT-Verbindungen mit AWS IoT Core abschließen kann. Sie können diese Einstellung anpassen, wenn die Operationen aufgrund von Bandbreitenbeschränkungen oder hoher Latenz zu einer Zeitüberschreitung führen. Um diese Einstellung in der Datei [config.json](#) zu konfigurieren, fügen Sie die `mqttOperationTimeout`-Eigenschaft im `coreThing`-Objekt hinzu, oder ändern Sie sie. Beispiel:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

Das Standard-Timeout beträgt 5 Sekunden. Der minimale Timeout beträgt 5 Sekunden.

Aktivieren der automatischen IP-Erkennung

Sie können so konfigurieren AWS IoT Greengrass , dass Client-Geräte in einer Greengrass-Gruppe den Greengrass-Core automatisch erkennen. Wenn diese Option aktiviert ist, beobachtet der Core Änderungen an seinen IP-Adressen. Wenn sich eine Adresse ändert, veröffentlicht der Core eine

aktualisierte Adressliste. Diese Adressen werden Client-Geräten zur Verfügung gestellt, die sich in derselben Greengrass-Gruppe wie der Core befinden.

 Note

Die AWS IoT Richtlinie für Client-Geräte muss den Geräten das `greengrass:Discover` Abrufen von Verbindungsinformationen für den Core gestatten. Weitere Informationen zur Richtlinienanweisung finden Sie unter [the section called "Discovery-Autorisierung"](#).

Um diese Funktion von der AWS IoT Greengrass Konsole aus zu aktivieren, wählen Sie Automatische Erkennung, wenn Sie Ihre Greengrass-Gruppe zum ersten Mal bereitstellen. Sie können diese Funktion auch auf der Gruppenkonfigurationsseite aktivieren oder deaktivieren, indem Sie die Registerkarte Lambda-Funktionen und dann den IP-Detektor auswählen. Die automatische IP-Erkennung ist aktiviert, wenn die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben ausgewählt ist.

Um die automatische Erkennung mit der AWS IoT Greengrass API zu verwalten, müssen Sie die Lambda-Funktion des `IPDetector` Systems konfigurieren. Das folgende Verfahren zeigt, wie Sie den [create-function-definition-version](#) CLI-Befehl verwenden, um die automatische Erkennung des Greengrass-Kerns zu konfigurieren.

1. Rufen Sie IDs die Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

 Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die LatestVersionWerte und Id aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.
 - Ersetzen Sie *group-id* durch den Id, den Sie kopiert haben.
 - Ersetzen Sie *latest-group-version-id* durch den LatestVersion, den Sie kopiert haben.

```
aws greengrass get-group-version \  
--group-id group-id \  
--group-version-id latest-group-version-id
```

4. Kopieren Sie aus dem Definition Objekt in der Ausgabe die CoreDefinitionVersionArn und die ARNs aller anderen Gruppenkomponenten außer FunctionDefinitionVersionArn. Diese Werte verwenden Sie, wenn Sie eine neue Gruppenversion erstellen.
5. Kopieren Sie die ID der Funktionsdefinition sowie die Funktionsdefinitionsversion aus FunctionDefinitionVersionArn in der Ausgabe:

```
arn:aws:greengrass:region:account-id:/greengrass/groups/function-definition-id/  
versions/function-definition-version-id
```

 Note

Sie können optional eine Funktionsdefinition mit dem [create-function-definition](#)-Befehl erstellen, und anschließend die ID aus der Ausgabe kopieren.

6. Verwenden Sie den Befehl [get-function-definition-version](#) zum Abrufen des aktuellen Definitionsstatus. Verwenden *function-definition-id* Sie das, was Sie kopiert haben, für die Funktionsdefinition. Beispiel, *4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*.

```
aws greengrass get-function-definition-version
```

```
--function-definition-id function-definition-id
--function-definition-version-id function-definition-version-id
```

Notieren Sie sich die aufgeführten Funktionskonfigurationen. Beim Erstellen einer neuen Funktionsdefinitionsversion müssen Sie diese angeben, um zu verhindern, dass Ihre aktuellen Definitionseinstellungen verloren gehen.

7. Fügen Sie der Funktionsdefinition eine Funktionsdefinitionsversion hinzu.

- *function-definition-id* Ersetzen Sie es durch Id das, was Sie für die Funktionsdefinition kopiert haben. Beispiel, *4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*.
- *arbitrary-function-id* Ersetzen Sie durch einen Namen für die Funktion, z. **auto-detection-function B**.
- Fügen Sie dem functions Array alle Lambda-Funktionen hinzu, die Sie in diese Version aufnehmen möchten, z. B. alle im vorherigen Schritt aufgeführten.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions
' [{"FunctionArn": "arn:aws:lambda:::function:GGIPDetector:1", "Id": "arbitrary-
function-id", "FunctionConfiguration":
{"Pinned": true, "MemorySize": 32768, "Timeout": 3}} ] '\
--region us-west-2
```

8. Kopieren Sie den Arn der Funktionsdefinitionsversion aus der Ausgabe.

9. Erstellen Sie eine Gruppenversion, die die Lambda-Funktion des Systems enthält.

- Ersetzen Sie *group-id* durch die Id für die Gruppe.
- *core-definition-version-arn* Ersetzen Sie es durch CoreDefinitionVersionArn das, was Sie aus der neuesten Gruppenversion kopiert haben.
- *function-definition-version-arn* Ersetzen Sie es durch Arn das, was Sie für die neue Version der Funktionsdefinition kopiert haben.
- Ersetzen Sie die ARNs für andere Gruppenkomponenten (z. B. SubscriptionDefinitionVersionArn oder DeviceDefinitionVersionArn), die Sie aus der neuesten Gruppenversion kopiert haben.
- Entfernen Sie alle nicht verwendeten Parameter. Entfernen Sie zum Beispiel --resource-definition-version-arn, wenn Ihre Gruppenversion keine Ressourcen enthält.

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

10. Kopieren Sie die Version aus der Ausgabe. Dies ist die ID der neuen Gruppenversion.

11. Stellen Sie die Gruppe mit der neuen Gruppenversion bereit.

- *group-id* Ersetzen Sie sie durch Id die, die Sie für die Gruppe kopiert haben.
- *group-version-id* Ersetzen Sie es durch Version das, was Sie für die neue Gruppenversion kopiert haben.

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

Wenn Sie die IP-Adresse Ihres Greengrass-Cores manuell eingeben möchten, können Sie dieses Tutorial mit einer anderen Funktionsdefinition abschließen, die die IPDetector-Funktion nicht enthält. Dadurch wird verhindert, dass die Erkennungsfunktion Ihre Greengrass-Core-IP-Adresse lokalisiert und automatisch eingibt.

Diese System-Lambda-Funktion ist in der Lambda-Konsole nicht sichtbar. Nachdem die Funktion der neuesten Gruppenversion hinzugefügt wurde, ist sie in Bereitstellungen enthalten, die Sie von der Konsole aus erstellen (es sei denn, Sie ersetzen oder verschieben sie mit der API).

Konfigurieren das Init-Systems zum Starten des Greengrass-Daemons

Es ist ein bewährtes Verfahren, Ihr Init-System so einzurichten, dass der Greengrass-Daemon beim Startvorgang ausgeführt wird, insbesondere bei der Verwaltung großer Geräteflotten.

 Note

Wenn Sie früher apt die AWS IoT Greengrass Core-Software installiert haben, können Sie die Systemd-Skripte verwenden, um den Start beim Systemstart zu aktivieren. Weitere Informationen finden Sie unter [the section called “Verwenden von systemd-Skripts zum Verwalten des Greengrass Daemon-Lebenszyklus”](#).

Es gibt verschiedene Arten von Init-Systemen, z. B. initd, systemd und SystemV, und sie verwenden ähnliche Konfigurationsparameter. Das folgende Beispiel ist eine Servicedatei für systemd. Der Type-Parameter wird auf `forking` festgelegt, da `greengrassd` (das zum Starten von Greengrass verwendet wird) den Greengrass-Daemon-Prozess verzweigt und der `Restart`-Parameter auf `on-failure` festgelegt wird, um systemd so zu steuern, dass Greengrass neu gestartet wird, wenn Greengrass einen fehlerhaften Zustand aufweist.

 Note

Führen Sie das `check_ggc_dependencies`-Skript wie in [Modul 1](#) beschrieben aus, um festzustellen, ob Ihr Gerät systemd verwendet. Zur Verwendung von systemd stellen Sie anschließend sicher, dass der `useSystemd`-Parameter in [config.json](#) auf `yes` festgelegt ist.

```
[Unit]
Description=Greengrass Daemon

[Service]
Type=forking
PIDFile=/var/run/greengrassd.pid
Restart=on-failure
ExecStart=/greengrass/ggc/core/greengrassd start
ExecReload=/greengrass/ggc/core/greengrassd restart
ExecStop=/greengrass/ggc/core/greengrassd stop

[Install]
WantedBy=multi-user.target
```

Weitere Informationen finden Sie auch unter

- [Was ist AWS IoT Greengrass?](#)
- [the section called “Unterstützte Plattformen und Anforderungen”](#)
- [Erste Schritte mit AWS IoT Greengrass](#)
- [the section called “Übersicht über das Gruppenobjektmodell”](#)
- [the section called “Integration von Hardware-Sicherheit”](#)

AWS IoT Greengrass Version 1 Wartungspolitik

Nutze diese AWS IoT Greengrass V1 Wartungsrichtlinie, um mehr über die verschiedenen Wartungs- und Aktualisierungsstufen für den AWS IoT Greengrass V1 Service und die Core-Software v1.x zu erfahren. AWS IoT Greengrass

Themen

- [AWS IoT Greengrass Versionsschema](#)
- [Lebenszyklusphasen für Hauptversionen der AWS IoT Greengrass Core-Software](#)
- [Wartungsrichtlinie für die AWS IoT Greengrass Core-Software](#)
- [Einstellungszeitplan](#)
- [Support-Richtlinie für AWS Lambda Funktionen auf Greengrass-Kerngeräten](#)
- [Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1](#)
- [Ende des Wartungsplans](#)

AWS IoT Greengrass Versionsschema

AWS IoT Greengrass verwendet [semantische Versionierung für die Core-Software](#). AWS IoT Greengrass Semantische Versionen folgen einer Hauptversion, geringfügig, Patch-Nummernsystem. Die Hauptversion wird für Funktions- und API-Änderungen erhöht, die nicht mit früheren Hauptversionen abwärtskompatibel sind. Die Nebenversion wird für Versionen erhöht, die neue abwärtskompatible Funktionen hinzufügen. Die Patch-Version wird für Sicherheitspatches oder Bugfixes inkrementiert. Seit der ersten Hauptversion, v1.0.0, AWS IoT Greengrass wurden 11 Nebenversionen der AWS IoT Greengrass Core-Software v1.x veröffentlicht, wobei v1.11.6 die neueste Version ist. Wir empfehlen Ihnen, Ihre AWS IoT Greengrass Core-Software auf die neueste verfügbare Version zu aktualisieren, um von neuen Funktionen, Verbesserungen und Bugfixes zu profitieren.

Im Dezember 2020 wurde das erste große Versionsupdate AWS IoT Greengrass veröffentlicht. Dieses Update beinhaltete den AWS IoT Greengrass V2 Service und die Version 2.0.3 der AWS IoT Greengrass Core-Software. Für neue Anwendungen empfehlen wir dringend, die AWS IoT Greengrass Core-Software AWS IoT Greengrass Version 2 v2.x zu verwenden. Version 2 erhält neue Funktionen, beinhaltet alle wichtigen V1-Funktionen und unterstützt zusätzliche Plattformen und kontinuierliche Bereitstellungen auf großen Geräteflotten. Weitere Informationen finden Sie unter [Was ist? AWS IoT Greengrass V2](#).

Lebenszyklusphasen für Hauptversionen der AWS IoT Greengrass Core-Software

Jede Hauptversion der AWS IoT Greengrass Core-Software hat die folgenden drei aufeinanderfolgenden Lebenszyklusphasen. Jede Lebenszyklusphase bietet unterschiedliche Wartungsstufen über einen bestimmten Zeitraum nach dem ursprünglichen Veröffentlichungsdatum.

- Veröffentlichungsphase — AWS IoT Greengrass möglicherweise werden die folgenden Updates veröffentlicht:
 - Kleinere Versionsupdates, die neue Funktionen oder Verbesserungen vorhandener Funktionen bieten
 - Patch-Versionsupdates, die Sicherheitspatches und Bugfixes bereitstellen
- Wartungsphase — AWS IoT Greengrass Möglicherweise werden Patch-Versionsupdates veröffentlicht, die Sicherheitspatches und Bugfixes enthalten. AWS IoT Greengrass veröffentlicht während der Wartungsphase keine neuen Funktionen oder Verbesserungen vorhandener Funktionen.
- Verlängerte Lebensphase — veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Fehlerkorrekturen bieten. Die AWS Cloud Endgeräte und API-Operationen bleiben jedoch weiterhin verfügbar und werden gemäß dem [AWS IoT Greengrass Service Level Agreement](#) betrieben. Geräte, auf denen die AWS IoT Greengrass Core-Software v1.x ausgeführt wird, können weiterhin eine Verbindung herstellen AWS Cloud und betrieben werden.

Nach Ablauf der erweiterten Lebensphase für eine Hauptversion von AWS IoT Greengrass sind die AWS Cloud Endgeräte und API-Operationen veraltet und nicht mehr verfügbar. Geräte, auf denen die AWS IoT Greengrass Core-Software v1.x ausgeführt wird, können zum Betrieb keine Verbindung zu AWS Cloud Diensten herstellen.

Wartungsrichtlinie für die AWS IoT Greengrass Core-Software

Die AWS IoT Greengrass Core-Software v1.x trat am 30. Juni 2023 in die erweiterte Lebensphase ein. Nach diesem Datum wird die AWS IoT Greengrass Core-Software v1.x bis auf weiteres in der erweiterten Lebensphase verbleiben.

Die AWS IoT Greengrass Core-Software v2.x befindet sich derzeit in der Release-Phase und wird bis auf weiteres in der Release-Phase bleiben. AWS IoT Greengrass fügt der AWS IoT Greengrass

Core-Software v2.x weiterhin neue Funktionen und Verbesserungen hinzu. Beispielsweise wurde die Windows-Unterstützung in Version 2.5.0 der Core-Software AWS IoT Greengrass veröffentlicht. AWS IoT Greengrass veröffentlicht Sicherheitspatches und Bugfixes für alle Nebenversionen von AWS IoT Greengrass Core v2.x für mindestens ein Jahr nach dem Veröffentlichungsdatum. Weitere Informationen finden Sie unter [Was ist neu in](#). AWS IoT Greengrass V2

Zeitplan für die Wartungsphase

Am 30. Juni 2023 endete die Wartungsphase für die AWS IoT Greengrass Core-Software v1.11.x. Am 31. März 2022 endete die Wartungsphase für die AWS IoT Greengrass Core-Software v1.10.x. Die Wartungsphase endet für bestimmte Artefakte und Funktionen der AWS IoT Greengrass Core-Software v1.x vor diesen Daten. Weitere Informationen finden Sie unter [Ende des Wartungsplans](#).

Wenn Sie einen AWS Support Plan haben, hat die Wartungsphase für AWS IoT Greengrass Core Software v1.x keinen Einfluss auf Ihren AWS Support Plan. Sie können auch nach Ablauf der Wartungsphase weiterhin AWS Support Tickets öffnen. Wenn Sie Fragen oder Bedenken haben, wenden Sie sich an Ihren AWS Support Ansprechpartner oder stellen Sie mithilfe des AWS IoT GreengrassTags eine Frage auf [AWS re:POST](#).

Einstellungszeitplan

Derzeit ist nicht geplant, die Unterstützung der AWS IoT Greengrass Core-Software v1.x einzustellen. Die AWS IoT Greengrass V1 Endpunkte und API-Operationen bleiben bis auf weiteres verfügbar. Die AWS IoT Greengrass Core-Software v1.11.6 trat am 30. Juni 2023 in die erweiterte Lebensphase ein. Während dieser Phase können Geräte, auf denen die AWS IoT Greengrass Core-Software v1.x ausgeführt wird, bis auf weiteres weiterhin eine Verbindung zum AWS IoT Greengrass V1 Dienst herstellen, um ihn zu betreiben.

Wenn AWS IoT Greengrass V1 der Support in future eingestellt AWS IoT Greengrass wird, werde ich Sie 12 Monate im Voraus darüber informieren, bevor dies geschieht. Dies hilft Ihnen bei der Planung der Aktualisierung Ihrer zu verwendenden Anwendungen AWS IoT Greengrass V2 und der AWS IoT Greengrass Core-Software v2.x. Weitere Informationen darüber, wie Sie Ihre Anwendungen auf Version 2 aktualisieren können, finden Sie unter [Umstieg von Version 2 AWS IoT Greengrass V1 auf Version 2](#).

Support-Richtlinie für AWS Lambda Funktionen auf Greengrass-Kerngeräten

AWS IoT Greengrass ermöglicht es Ihnen, AWS Lambda Funktionen auf IoT-Geräten auszuführen. AWS Lambda bietet eine Supportrichtlinie und Zeitpläne, die die Unterstützung für Lambda-Laufzeiten in festlegen. AWS IoT Greengrass Wenn eine Lambda-Laufzeit das Ende der Supportphase erreicht hat, wird AWS IoT Greengrass auch der Support für diese Laufzeit beendet. Weitere Informationen finden Sie unter [Runtime-Supportrichtlinie](#) im AWS Lambda Entwicklerhandbuch.

Wenn eine Lambda-Laufzeit das Ende des Supports erreicht, können Sie keine Lambda-Funktionen erstellen oder aktualisieren, die diese Laufzeit verwenden. Sie können diese Lambda-Funktionen jedoch weiterhin auf Greengrass-Kerngeräten bereitstellen und bereitgestellte Lambda-Funktionen aufrufen. Diese Richtlinie gilt auch für AWS IoT Greengrass V2

Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) für AWS IoT Greengrass V1 ermöglicht es Ihnen, Ihre AWS IoT Greengrass Geräte für die Aufnahme in den [AWS Partner Gerätecatalog](#) zu validieren und zu [qualifizieren](#). Ab dem 4. April 2022 generiert AWS IoT Device Tester (IDT) für AWS IoT Greengrass V1 keine signierten Qualifikationsberichte mehr. Sie können neue AWS IoT Greengrass V1 Geräte nicht mehr über das [Gerätequalifizierungsprogramm für die AWS Partner Aufnahme in den AWS Gerätecatalog qualifizieren](#). Sie können Greengrass V1-Geräte zwar nicht qualifizieren, aber Sie können IDT weiterhin verwenden, um Ihre Greengrass V1-Geräte AWS IoT Greengrass V1 zu testen. Wir empfehlen Ihnen, [IDT for AWS IoT Greengrass V2](#) zu verwenden, um Greengrass-Geräte zu qualifizieren und im [AWS Partner Gerätecatalog](#) aufzulisten. Weitere Informationen finden Sie unter [Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1](#).

Ende des Wartungsplans

In der folgenden Tabelle sind die Termine für das Ende der Wartungsarbeiten für Artefakte und Funktionen von AWS IoT Greengrass Core v1.x aufgeführt. Wenn Sie Fragen zum Wartungsplan oder zu den Wartungsrichtlinien haben, wenden Sie sich an den [AWS Support](#).

Artifact oder Merkmal	Ende des Wartungsdatums
Installation des Greengrass APT-Repositorys	11. Februar 2022
Anschluss zur ML-Bildklassifizierung	31. März 2022
Anschluss zur ML-Objekterkennung	31. März 2022
ML-Feedback-Anschluss	31. März 2022
AWS IoT Analytics Anschluss	31. März 2022
Konnektor für Twilio-Benachrichtigungen	31. März 2022
Konnektor für die Splunk-Integration	31. März 2022
Serieller Stream-Anschluss	31. März 2022
ServiceNow MetricBase Integrationsanschluss	31. März 2022
Raspberry Pi GPIO-Anschluss	31. März 2022
AWS IoT Greengrass Kernsoftware v1.10.x	31. März 2022
AWS IoT Greengrass Kernsoftware v1.x Docker-Images	30. Juni 2022
AWS IoT Greengrass Kernsoftware v1.11.x	30. Juni 2023
AWS IoT Greengrass Kernsoftware v1.11.x Snap	31. Dezember 2023

Ende der Wartung für AWS IoT Greengrass Docker-Images der Core-Software v1.x

Am 30. Juni 2022 AWS IoT Greengrass wurde die Wartung für Docker-Images der AWS IoT Greengrass Core-Software v1.x eingestellt, die in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub veröffentlicht wurden. Sie können diese Docker-Images weiterhin bis zum 30. Juni 2023, also 1 Jahr nach Ende der Wartung, von Amazon ECR und Docker Hub

herunterladen. Nach dem Ende der Wartung am 30. Juni 2022 erhalten die Docker-Images der AWS IoT Greengrass Core-Software v1.x jedoch keine Sicherheitspatches oder Bugfixes mehr. Wenn Sie einen Produktions-Workload ausführen, der von diesen Docker-Images abhängt, empfehlen wir Ihnen, Ihre eigenen Docker-Images mithilfe der bereitgestellten Dockerfiles zu erstellen. AWS IoT Greengrass Weitere Informationen finden Sie unter [AWS IoT Greengrass Docker-Software](#).

Ende der Wartung für das APT-Repository der AWS IoT Greengrass Core-Software v1.x

Am 11. Februar 2022 AWS IoT Greengrass endete die Wartung für die Option, [die AWS IoT Greengrass Core-Software v1.x aus einem APT-Repository zu installieren](#). Das APT-Repository wurde an diesem Tag entfernt, sodass Sie das APT-Repository nicht mehr verwenden können, um die AWS IoT Greengrass Core-Software zu aktualisieren oder die AWS IoT Greengrass Core-Software auf neuen Geräten zu installieren. Auf Geräten, auf denen Sie das AWS IoT Greengrass Repository hinzugefügt haben, müssen Sie [das Repository aus der Quellenliste entfernen](#). Wir empfehlen, dass Sie die AWS IoT Greengrass Core-Software v1.x mithilfe von [Tar-Dateien](#) aktualisieren.

Ende der Wartung für die AWS IoT Greengrass Core-Software v1.11.x Snap

[Am 31. Dezember 2023 AWS IoT Greengrass endet die Wartung für die AWS IoT Greengrass Kernsoftwareversion 1.11.x Snap, die auf snapcraft.io veröffentlicht wurde](#). Geräte, auf denen der Snap derzeit ausgeführt wird, funktionieren bis auf weiteres weiter. Nach Ende der Wartung erhält der AWS IoT Greengrass Core-Snap jedoch keine Sicherheitspatches oder Bugfixes mehr.

Erste Schritte mit AWS IoT Greengrass

Dieses Tutorial „Erste Schritte“ umfasst mehrere Module, die Ihnen die AWS IoT Greengrass Grundlagen vermitteln und Ihnen bei den ersten Schritten helfen sollen AWS IoT Greengrass. Dieses Tutorial behandelt grundlegende Konzepte wie beispielsweise:

- Konfiguration von AWS IoT Greengrass Kernen und Gruppen.
- Der Bereitstellungsprozess für die Ausführung von AWS Lambda Funktionen am Edge.
- AWS IoT Geräte, sogenannte Client-Geräte, mit dem AWS IoT Greengrass Core verbinden.
- Erstellen von Abonnements, um die MQTT-Kommunikation zwischen lokalen Lambda-Funktionen, Client-Geräten und zu ermöglichen. AWS IoT

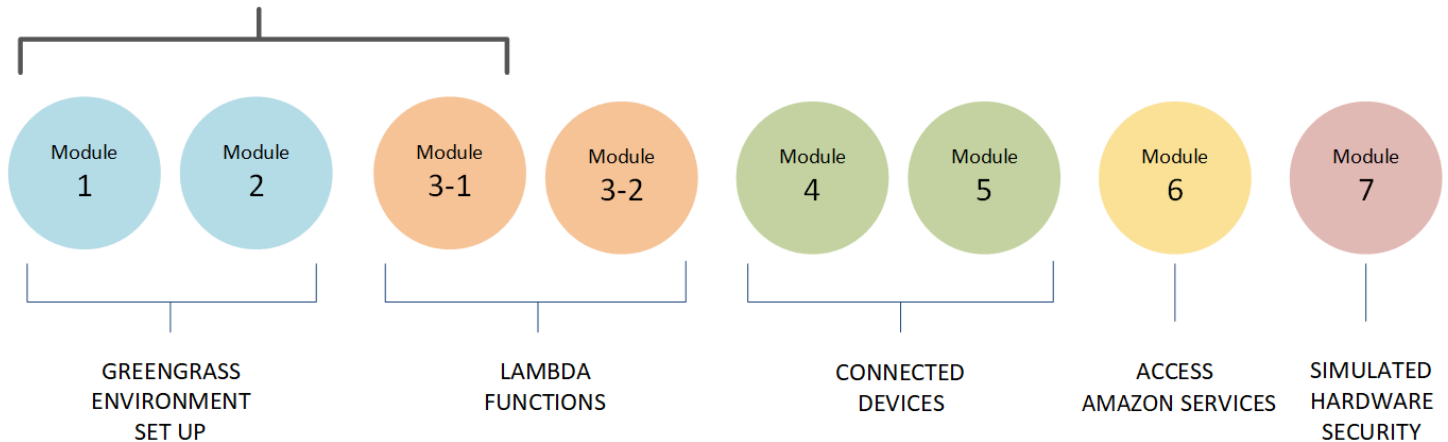
Wählen Sie aus, wie Sie beginnen möchten AWS IoT Greengrass

Sie können wählen, wie Sie dieses Tutorial verwenden, um Ihr Core-Gerät einzurichten:

- Führen Sie das [Greengrass-Geräte-Setup](#) auf Ihrem Kerngerät aus, sodass Sie in wenigen Minuten von der Installation von AWS IoT Greengrass Abhängigkeiten bis zum Testen einer Hello World Lambda-Funktion fertig sind. Dieses Skript reproduziert die Schritte in Modul 1 bis Modul 3-1.

– oder –

- Gehen Sie die Schritte in Modul 1 bis Modul 3-1 durch, um sich die Anforderungen und Prozesse von Greengrass genauer anzusehen. Mit diesen Schritten richten Sie Ihr Kerngerät ein, erstellen und konfigurieren eine Greengrass-Gruppe, die eine Hello World Lambda-Funktion enthält, und stellen Ihre Greengrass-Gruppe bereit. In der Regel nimmt dies eine oder zwei Stunden in Anspruch.

Quick Start: Greengrass Device Setup**Schnellstart**

[Greengrass Device Setup](#) konfiguriert Ihr Core-Gerät und Ihre Greengrass-Ressourcen. Das Skript:

- Installiert Abhängigkeiten. AWS IoT Greengrass
- Lädt das CA-Stammzertifikat sowie das Zertifikat und die Schlüssel für das Core-Gerät herunter.
- Lädt die AWS IoT Greengrass Core-Software auf Ihr Gerät herunter, installiert und konfiguriert sie.
- Startet den Greengrass-Daemon-Prozess auf dem Core-Gerät.
- Erstellt oder aktualisiert die [Greengrass-Servicerolle](#), falls erforderlich.
- Erstellt eine Greengrass-Gruppe und ein Greengrass Core.
- (Optional) Erstellt eine Hello World Lambda-Funktions-, Abonnement- und lokale Protokollierungskonfiguration.
- (Optional) Stellt die Greengrass-Gruppe bereit.

Module 1 und 2

[Modul 1](#) und [Modul 2](#) beschreiben, wie Sie Ihre Umgebung einrichten. (Oder verwenden Sie [Greengrass Device Setup](#), um diese Module für Sie auszuführen.)

- Konfigurieren Sie Ihr Core-Gerät für Greengrass.
- führen Sie das Skript für die Abhängigkeitsprüfung aus
- Erstellen Sie eine Greengrass-Gruppe und ein Greengrass Core.

- Laden Sie die neueste AWS IoT Greengrass Core-Software aus einer Datei tar.gz herunter und installieren Sie sie.
- Starten Sie den Greengrass-Daemon-Prozess auf dem Core.

 Note

AWS IoT Greengrass bietet auch andere Optionen für die Installation der AWS IoT Greengrass Core-Software, einschließlich apt Installationen auf unterstützten Debian-Plattformen. Weitere Informationen finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

Module 3-1 und 3-2

[Modul 3-1](#) und [Modul 3-2](#) beschreiben, wie lokale Lambda-Funktionen verwendet werden. (Oder verwenden Sie [Greengrass Device Setup](#), um Modul 3-1 für Sie auszuführen.)

- Erstellen Sie Hello World Lambda-Funktionen in AWS Lambda.
- Fügen Sie Ihrer Greengrass-Gruppe Lambda-Funktionen hinzu.
- Erstellen Sie Abonnements, die die MQTT-Kommunikation zwischen den Lambda-Funktionen und ermöglichen. AWS IoT
- Konfigurieren Sie die lokale Protokollierung für Greengrass-Systemkomponenten und Lambda-Funktionen.
- Stellen Sie eine Greengrass-Gruppe bereit, die Ihre Lambda-Funktionen und Abonnements enthält.
- Senden Sie Nachrichten von lokalen Lambda-Funktionen an AWS IoT.
- Rufen Sie lokale Lambda-Funktionen von auf. AWS IoT
- Testen Sie On-Demand- und langlebige Funktionen.

Module 4 und 5

[Modul 4](#) zeigt, wie Client-Geräte eine Verbindung zum Core herstellen und miteinander kommunizieren.

[Modul 5](#) zeigt, wie Client-Geräte Schatten verwenden können, um den Status zu kontrollieren.

- AWS IoT Geräte registrieren und bereitstellen (dargestellt durch Befehlszeilenterminals).

- Installieren Sie das AWS IoT Device SDK für Python. Dies wird von Client-Geräten verwendet, um den Greengrass-Kern zu entdecken.
- Fügen Sie die Client-Geräte zu Ihrer Greengrass-Gruppe hinzu.
- Erstellen Sie Abonnements, die MQTT-Kommunikation ermöglichen.
- Stellen Sie eine Greengrass-Gruppe bereit, die Ihre Client-Geräte enthält.
- Testen Sie die device-to-device Kommunikation.
- Testen Sie die Aktualisierungen des Schattenstatus.

Modul 6

[Modul 6](#) zeigt Ihnen, wie Lambda-Funktionen auf die AWS Cloud zugreifen können.

- Erstellen Sie eine Greengrass-Gruppenrolle, die den Zugriff auf Amazon DynamoDB DynamoDB-Ressourcen ermöglicht.
- Fügen Sie Ihrer Greengrass-Gruppe eine Lambda-Funktion hinzu. Diese Funktion verwendet das AWS SDK für Python, um mit DynamoDB zu interagieren.
- Erstellen Sie Abonnements, die MQTT-Kommunikation ermöglichen.
- Testen Sie die Interaktion mit DynamoDB.

Modul 7

[Modul 7](#) zeigt Ihnen, wie Sie ein simuliertes Hardware-Sicherheitsmodul (HSM) für die Verwendung mit einem Greengrass Core konfigurieren.

Important

Dieses erweiterte Modul wird nur für Experimente und erste Tests bereitgestellt. Es ist nicht für Produktionszwecke geeignet.

- Installieren und konfigurieren Sie ein softwarebasiertes HSM und einen privaten Schlüssel.
- Konfigurieren Sie den Greengrass Core für die Verwendung von Hardwaresicherheit.
- Testen Sie die Hardware-Sicherheitskonfiguration.

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein PC, Mac-, Windows- oder UNIX-ähnliches System.
- Ein AWS-Konto Falls Sie noch keines haben, beachten Sie die Informationen unter [the section called “Erstelle eine AWS-Konto”](#).
- Die Verwendung einer AWS [Region](#), die unterstützt AWS IoT Greengrass. Eine Liste der unterstützten Regionen für AWS IoT Greengrass finden Sie unter [AWS Endpunkte und Kontingente](#) in der Allgemeine AWS-Referenz.

Note

Notieren Sie sich Ihre AWS-Region und stellen Sie sicher, dass sie in diesem Tutorial einheitlich verwendet wird. Wenn Sie Ihre AWS-Region während des Tutorials wechseln, können Probleme beim Ausführen der Schritte auftreten.

- Ein Raspberry Pi 4 Model B oder Raspberry Pi 3 Model B/B+ mit einer 8-GB-microSD-Karte oder eine Amazon-Instance. EC2 Da AWS IoT Greengrass mit physischer Hardware verwendet werden sollte, empfehlen wir die Verwendung eines Raspberry Pi.

Note

Führen Sie den folgenden Befehl aus, um das Modell Ihres Raspberry Pi zu bestimmen:

```
cat /proc/cpuinfo
```

Merken Sie sich den Wert des Revision-Attributs am Ende der Liste und konsultieren Sie die Tabelle [Which Pi have I got?](#). Beispiel: Wenn der Wert für Revision a02082 lautet, geht aus der Tabelle hervor, dass es sich um einen Pi 3 Model B handelt.

Führen Sie den folgenden Befehl aus, um die Architektur Ihres Raspberry Pi zu bestimmen:

```
uname -m
```

In diesem Tutorial sollte das Ergebnis größer oder gleich armv71 sein.

- Grundlegende Vertrautheit mit Python.

Dieses Tutorial ist zwar für die Ausführung AWS IoT Greengrass auf einem Raspberry Pi vorgesehen, unterstützt AWS IoT Greengrass aber auch andere Plattformen. Weitere Informationen finden Sie unter [the section called “Unterstützte Plattformen und Anforderungen”](#).

Erstelle eine AWS-Konto

Wenn Sie noch keine haben AWS-Konto, gehen Sie wie folgt vor, um eine zu erstellen und zu aktivieren AWS-Konto:

Melde dich an für ein AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie [https://portal.aws.amazon.com/billing/die Anmeldung](https://portal.aws.amazon.com/billing/die-Anmeldung).
2. Folgen Sie den Online-Anweisungen.

Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontos wird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Administratorbenutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um [Aufgaben auszuführen, die Root-Benutzerzugriff erfordern](#).

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu <https://aws.amazon.com/> gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich [AWS-Managementkonsole](#) als Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter [Anmelden als Root-Benutzer](#) im AWS-Anmeldung Benutzerhandbuch zu.

2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

Anweisungen finden Sie unter [Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer \(Konsole\)](#) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

Anweisungen finden Sie unter [Aktivieren AWS IAM Identity Center](#) im AWS IAM Identity Center Benutzerhandbuch.

2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter [Benutzerzugriff mit der Standardeinstellung konfigurieren](#).AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie [im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal](#).

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

Anweisungen hierzu finden Sie unter [Berechtigungssatz erstellen](#) im AWS IAM Identity Center Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

Eine genaue Anleitung finden Sie unter [Gruppen hinzufügen](#) im AWS IAM Identity Center Benutzerhandbuch.

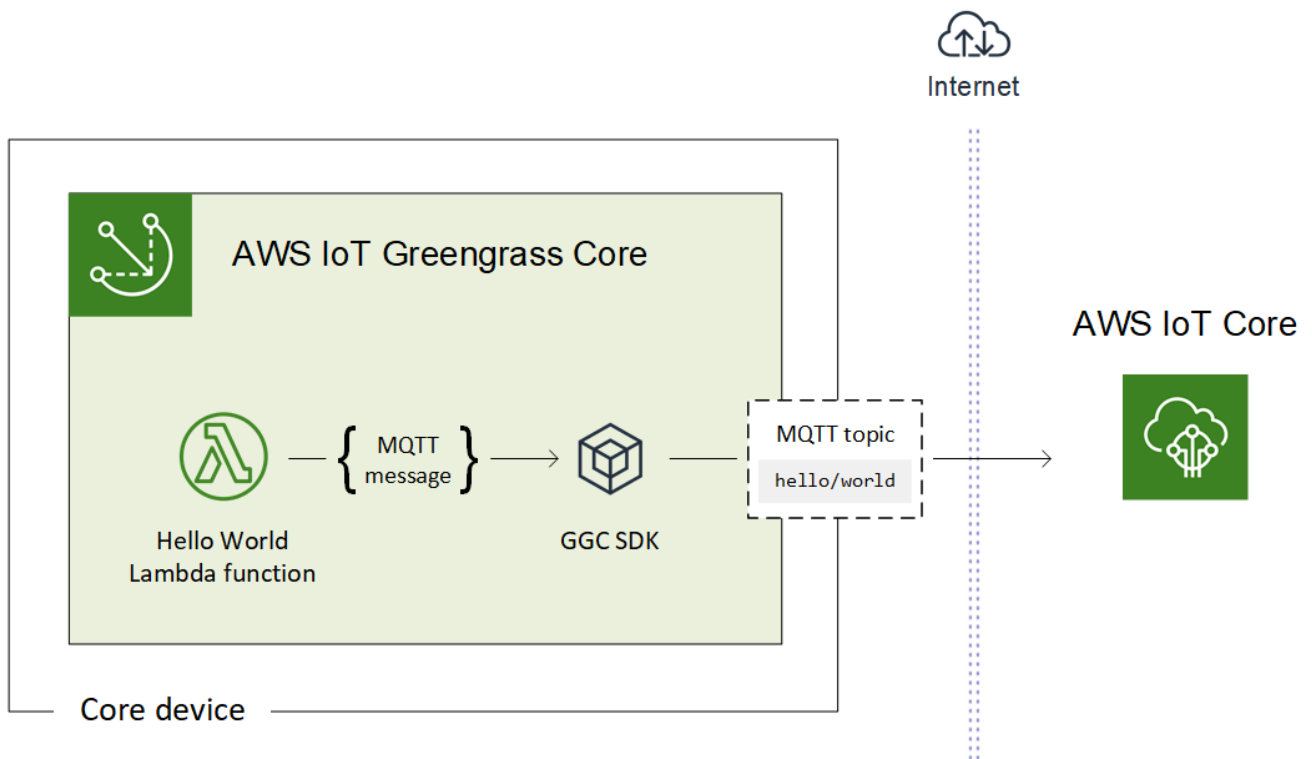
⚠ Important

Für dieses Tutorial gehen wir davon aus, dass Ihr IAM-Benutzerkonto über Administratorzugriffsberechtigungen verfügt.

Schnellstart: Greengrass-Geräteeinrichtung

Das Greengrass-Geräte-Setup ist ein Skript, das Ihr Kerngerät in wenigen Minuten einrichtet, sodass Sie es verwenden AWS IoT Greengrass können. Verwenden Sie dieses Skript, um:

1. Konfigurieren Sie Ihr Gerät und installieren Sie die AWS IoT Greengrass Core-Software.
2. Konfigurieren Sie Ihre Cloud-basierten Ressourcen.
3. Stellen Sie optional eine Greengrass-Gruppe mit einer Hello World Lambda-Funktion bereit, an die MQTT-Nachrichten vom Core AWS IoT aus gesendet werden. AWS IoT Greengrass Dadurch wird die in der folgenden Abbildung gezeigte Greengrass-Umgebung eingerichtet.



Voraussetzungen

Für das Greengrass Device Setup gelten die folgenden Anforderungen:

- Ihr Core-Gerät muss eine [unterstützte Plattform](#) verwenden. Auf dem Gerät muss ein entsprechender Paketmanager installiert sein: apt, yum oder opkg.
- Der Linux-Benutzer, der das Skript ausführt, muss über Berechtigungen zum Ausführen als sudo verfügen.
- Sie müssen Ihre AWS-Konto Anmeldeinformationen angeben. Weitere Informationen finden Sie unter [the section called “Geben Sie die Anmeldeinformationen ein AWS-Konto”](#).

Note

Das Greengrass-Geräte-Setup installiert die [neueste Version](#) der AWS IoT Greengrass Core-Software auf dem Gerät. Durch die Installation der AWS IoT Greengrass Core-Software stimmen Sie der [Greengrass Core-Softwarelizenzvereinbarung](#) zu.

Ausführen von Greengrass Device Setup

Sie können Greengrass Device Setup in nur wenigen Schritten ausführen. Nachdem Sie Ihre AWS-Konto Anmeldeinformationen eingegeben haben, stellt das Skript Ihr Greengrass-Core-Gerät bereit und stellt innerhalb von Minuten eine Greengrass-Gruppe bereit. Führen Sie die folgenden Befehle in einem Terminalfenster auf dem Zielgerät aus.

Note

Diese folgenden Schritte zeigen, wie Sie das Skript im interaktiven Modus ausführen, in dem Sie aufgefordert werden, die einzelnen Eingabewerte einzugeben oder zu akzeptieren. Informationen zum Ausführen des Skripts im Hintergrund finden Sie unter [the section called “Ausführen von Greengrass Device Setup im Hintergrundmodus”](#).

1. [Stellen Sie Ihre Anmeldeinformationen bereit](#). In diesem Verfahren gehen wir davon aus, dass Sie temporäre Sicherheitsanmeldeinformationen als Umgebungsvariablen angeben.

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

 Note

Wenn Sie das Greengrass-Geräte-Setup auf einem Raspbian oder einer OpenWrt Plattform ausführen, erstellen Sie eine Kopie dieser Befehle. Sie müssen sie nach dem Neustart des Geräts erneut angeben.

2. Laden Sie das Skript herunter und starten Sie es. Sie können `wget` oder `curl` verwenden, um das Skript herunterzuladen.

`wget`:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
greengrass-interactive
```

`curl`:

```
curl https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/
gg-device-setup-latest.sh > gg-device-setup-latest.sh && chmod +x ./gg-device-
setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-
interactive
```

3. Fahren Sie mit den Eingabeaufforderungen für [Eingabewerte](#) fort. Sie können die Eingabetaste drücken, um den Standardwert zu verwenden, oder einen benutzerdefinierten Wert eingeben und dann die Eingabetaste drücken.

Das Skript schreibt Statusmeldungen an das Terminal, die in etwa wie folgt aussehen.


```
##### Greengrass Device Setup v1.0.0 #####
[GreengrassDeviceSetup] The Greengrass Device Setup bootstrap log is available at: /tmp/greengrass-device-setup-bootstrap-1575933831.log
[GreengrassDeviceSetup] Using package management tool: yum...
[GreengrassDeviceSetup] Using runtime: python3.7...
[GreengrassDeviceSetup] Installing a dedicated pip for Greengrass Device Setup...
[GreengrassDeviceSetup] Validating and installing required dependencies...
[GreengrassDeviceSetup] The Greengrass Device Setup configuration is complete. Starting the Greengrass environment setup...
[GreengrassDeviceSetup] Forwarding command-line parameters: bootstrap-greengrass-interactive

[GreengrassDeviceSetup] Validating the device environment...
[GreengrassDeviceSetup] Validation of the device environment is complete.

[GreengrassDeviceSetup] Running the Greengrass environment setup...
[GreengrassDeviceSetup] The Greengrass environment setup is complete.

[GreengrassDeviceSetup] Configuring cloud-based Greengrass group management...
[GreengrassDeviceSetup] The Greengrass group configuration is complete.

[GreengrassDeviceSetup] Preparing the Greengrass core software...
[GreengrassDeviceSetup] The Greengrass core software is running.

[GreengrassDeviceSetup] Configuring the group deployment...
[GreengrassDeviceSetup] The group deployment is complete.
```

4. Wenn auf Ihrem Kerngerät Raspbian oder ausgeführt wird OpenWrt, starten Sie das Gerät neu, wenn Sie dazu aufgefordert werden, geben Sie Ihre Anmeldeinformationen ein und starten Sie dann das Skript neu.
 - a. Wenn Sie zum Neustart des Geräts aufgefordert werden, führen Sie einen der folgenden Befehle aus.

Für Raspbian Plattformen:

```
sudo reboot
```

Für OpenWrt Plattformen:

```
reboot
```

- b. Öffnen Sie nach dem Neustart des Geräts das Terminal und geben Sie Ihre Anmeldeinformationen als Umgebungsvariablen an.

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

- c. Starten Sie das Skript erneut.

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

- d. Wenn Sie aufgefordert werden, die Eingabewerte aus der vorherigen Sitzung zu verwenden oder eine neue Installation zu starten, geben Sie `yes` ein, um Ihre Eingabewerte wiederzuverwenden.

 Note

Auf Plattformen, für die ein Neustart erforderlich ist, werden Ihre Eingabewerte (mit Ausnahme der Anmeldeinformationen) der vorherigen Sitzung vorübergehend in der Datei `GreengrassDeviceSetup.config.info` gespeichert.

Wenn die Einrichtung abgeschlossen ist, zeigt das Terminal eine Erfolgsstatusmeldung an, die in etwa wie folgt aussieht.

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers1
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mu11v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

=====
```

5. Sehen Sie sich die neue Greengrass-Gruppe an, die das Skript anhand der von Ihnen angegebenen Eingabewerte konfiguriert.
 - a. Melden Sie sich [AWS-Managementkonsole](#) auf Ihrem Computer an und öffnen Sie die AWS IoT Konsole.

 Note

Vergewissern Sie sich, dass es sich bei der AWS-Region Auswahl in der Konsole um dieselbe handelt, die Sie zur Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

- b. Erweitern Sie im Navigationsbereich Greengrass-Geräte und wählen Sie dann Gruppen (V1), um die neu erstellte Gruppe zu suchen.
6. Wenn Sie die Hello World Lambda-Funktion hinzugefügt haben, stellt das Greengrass-Geräte-Setup die Greengrass-Gruppe auf Ihrem Kerngerät bereit. Um die Lambda-Funktion zu testen oder Informationen darüber zu erhalten, wie Sie die Lambda-Funktion aus der Gruppe entfernen können, fahren Sie mit Modul 3-1 des Tutorials Erste Schritte fort. [the section called “Stellen Sie sicher, dass die Lambda-Funktion auf dem Kerngerät ausgeführt wird”](#)

 Note

Vergewissern Sie sich, dass es sich bei der AWS-Region Auswahl in der Konsole um dieselbe handelt, die Sie zur Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

Wenn Sie die Hello World Lambda-Funktion nicht enthalten haben, können Sie [Ihre eigene Lambda-Funktion erstellen](#) oder andere Greengrass-Funktionen ausprobieren. Beispielsweise können Sie den [Docker-Anwendungsbereitstellungs](#)-Konnektor zu Ihrer Gruppe hinzufügen und verwenden, um Docker-Container auf Ihrem Core-Gerät bereitzustellen.

Beheben von -Problemen

Sie können die folgenden Informationen verwenden, um Probleme mit der AWS IoT Greengrass Geräteeinrichtung zu beheben.

Fehler: Python (Python3.7) wurde nicht gefunden. Es wird versucht, es zu installieren...

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn Sie mit einer EC2 Amazon-Instance arbeiten. Dieser Fehler tritt auf, wenn Python nicht im `/usr/bin/python3.7` Ordner installiert

ist. Um diesen Fehler zu beheben, verschieben Sie Python nach der Installation in das richtige Verzeichnis:

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

Zusätzliche Problembehebung

Um weitere Probleme mit dem AWS IoT Greengrass Geräte-Setup zu beheben, können Sie in den Protokolldateien nach Debug-Informationen suchen:

- Überprüfen Sie bei Problemen mit der Konfiguration von Greengrass Device Setup die Datei /tmp/greengrass-device-setup-bootstrap-*epoch-timestamp*.log.
- Überprüfen Sie bei Problemen mit der Einrichtung der Greengrass-Gruppe oder der Core-Umgebung die Datei GreengrassDeviceSetup-*date-time*.log im selben Verzeichnis wie gg-device-setup-latest.sh oder an dem angegebenen Speicherort.

Weitere Hilfe zur Fehlerbehebung findest du unter [AWS IoT Greengrass dem Tag auf AWS re:POST Fehlerbehebung oder überprüfe das Tag](#).

Konfigurationsoptionen für Greengrass Device Setup

Sie konfigurieren das Greengrass-Geräte-Setup für den Zugriff auf Ihre AWS Ressourcen und richten Ihre Greengrass-Umgebung ein.

Geben Sie die Anmeldeinformationen ein AWS-Konto

Das Greengrass-Geräte-Setup verwendet Ihre AWS-Konto Anmeldeinformationen, um auf Ihre AWS Ressourcen zuzugreifen. Es unterstützt langfristige Anmeldeinformationen für einen IAM-Benutzer oder temporäre Sicherheitsanmeldeinformationen für eine IAM-Rolle.

Rufen Sie zunächst Ihre Anmeldeinformationen ab.

- Um langfristige Anmeldeinformationen zu verwenden, geben Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel für Ihren IAM-Benutzer an. Informationen zum Erstellen von Zugriffsschlüsseln für langfristige Anmeldeinformationen finden Sie unter [Verwaltung von Zugriffsschlüsseln für IAM-Benutzer](#) im IAM-Benutzerhandbuch.

- Um temporäre Sicherheitsanmeldeinformationen zu verwenden (empfohlen), geben Sie die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungstoken einer angenommenen IAM-Rolle an. Informationen zum Extrahieren temporärer Sicherheitsanmeldedaten aus dem AWS STS `assume-role` Befehl finden Sie unter [Verwenden temporärer Sicherheitsanmeldeinformationen mit dem AWS CLI](#) im IAM-Benutzerhandbuch.

Note

Für die Zwecke dieses Tutorials gehen wir davon aus, dass der IAM-Benutzer oder die IAM-Rolle über Administratorzugriffsberechtigungen verfügt.

Geben Sie dann Ihre Anmeldeinformationen für Greengrass Device Setup auf eine der beiden folgenden Arten an:

- Als Umgebungsvariablen. Legen Sie die Umgebungsvariablen `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` und `AWS_SESSION_TOKEN` (falls erforderlich) wie in Schritt 1 von [the section called “Ausführen von Greengrass Device Setup”](#) gezeigt fest, bevor Sie das Skript starten,
- Als Eingabewerte. Geben Sie nach dem Start des Skripts die Zugriffsschlüssel-ID, den geheimen Zugriffsschlüssel und das Sitzungs-Token (falls erforderlich) direkt im Terminal ein.

Greengrass Device Setup speichert Ihre Anmeldeinformationen nicht.

Bereitstellen von Eingabewerten

Im interaktiven Modus fordert Greengrass Device Setup Sie zum Bereitstellen von Eingabewerten auf. Sie können die Eingabetaste drücken, um den Standardwert zu verwenden, oder einen benutzerdefinierten Wert eingeben und dann die Eingabetaste drücken. Im Hintergrundmodus geben Sie nach dem Start des Skripts Eingabewerte ein.

Eingabewerte

AWS Zugriffsschlüssel-ID

Die Zugriffsschlüssel-ID aus den langfristigen oder temporären Anmeldeinformationen. Geben Sie diese Option nur als Eingabewert an, wenn Sie Ihre Anmeldeinformationen nicht als Umgebungsvariablen angeben. Weitere Informationen finden Sie unter [the section called “Geben Sie die Anmeldeinformationen ein AWS-Konto”](#).

Optionsname für den Hintergrundmodus: `--aws-access-key-id`

AWS Geheimer Zugriffsschlüssel

Der geheime Zugriffsschlüssel aus den langfristigen oder temporären Anmeldeinformationen. Geben Sie diese Option nur als Eingabewert an, wenn Sie Ihre Anmeldeinformationen nicht als Umgebungsvariablen angeben. Weitere Informationen finden Sie unter [the section called “Geben Sie die Anmeldeinformationen ein AWS-Konto”](#).

Optionsname für den Hintergrundmodus: `--aws-secret-access-key`

AWS Sitzungstoken

Das Sitzungs-Token aus den temporären Sicherheitsanmeldeinformationen. Geben Sie diese Option nur als Eingabewert an, wenn Sie Ihre Anmeldeinformationen nicht als Umgebungsvariablen angeben. Weitere Informationen finden Sie unter [the section called “Geben Sie die Anmeldeinformationen ein AWS-Konto”](#).

Optionsname für den Hintergrundmodus: `--aws-session-token`

AWS-Region

Der AWS-Region Ort, an dem Sie die Greengrass-Gruppe erstellen möchten. Eine Liste der unterstützten AWS-Region s finden Sie [AWS IoT Greengrass](#) in der Allgemeine Amazon Web Services-Referenz.

Standardwert: `us-west-2`

Optionsname für den Hintergrundmodus: `--region`

Group name (Gruppenname)

Der Name für die Greengrass-Gruppe.

Standardwert: `GreengrassDeviceSetup_Group_`*guid*

Optionsname für den Hintergrundmodus: `--group-name`

Core-Name

Der Name für den Greengrass Core. Der Kern ist ein AWS IoT Gerät (Ding), auf dem die AWS IoT Greengrass Core-Software ausgeführt wird. Der Kern wird der AWS IoT Registrierung und der Greengrass-Gruppe hinzugefügt. Wenn Sie einen Namen angeben, muss dieser im AWS-Konto und AWS-Region eindeutig sein.

Standardwert: `GreengrassDeviceSetup_Core_`*guid*

Optionsname für den Hintergrundmodus: `--core-name`

AWS IoT Greengrass Installationspfad der Kernsoftware

Der Speicherort im Gerätedateisystem, an dem Sie die AWS IoT Greengrass Core-Software installieren möchten.

Standardwert: `/`

Optionsname für den Hintergrundmodus: `--ggc-root-path`

"Hello World"-Lambda-Funktion

Gibt an, ob eine Hello World Lambda-Funktion in die Greengrass-Gruppe aufgenommen werden soll. Die Funktion veröffentlicht alle fünf Sekunden eine MQTT-Nachricht an das `hello/world`-Thema.

Das Skript erstellt und veröffentlicht diese benutzerdefinierte Lambda-Funktion in AWS Lambda und fügt sie Ihrer Greengrass-Gruppe hinzu. Das Skript erstellt darüber hinaus ein Abonnement in der Gruppe, das es der Funktion ermöglicht, MQTT-Nachrichten an AWS IoT zu senden.

Note

Dies ist eine Lambda-Funktion von Python 3.7. Wenn Python 3.7 nicht auf dem Gerät installiert ist und vom Skript nicht installiert werden kann, gibt das Skript eine Fehlermeldung im Terminal aus. Um die Lambda-Funktion in die Gruppe aufzunehmen, müssen Sie Python 3.7 manuell installieren und das Skript neu starten. Um die Greengrass-Gruppe ohne die Lambda-Funktion zu erstellen, starten Sie das Skript neu und geben Sie ein, `no` wenn Sie aufgefordert werden, die Funktion einzubeziehen.

Standardwert: `no`

Optionsname für den Hintergrundmodus: `--hello-world-lambda` – Diese Option akzeptiert keinen Wert. Fügen Sie sie in Ihren Befehl ein, wenn Sie die Funktion erstellen möchten.

Timeout bei der Bereitstellung

Die Anzahl der Sekunden, bevor Greengrass Device Setup die Überprüfung des Status der [Greengrass-Gruppenbereitstellung](#) beendet. Dies wird nur verwendet, wenn die Gruppe die "Hello World"-Lambda-Funktion enthält. Andernfalls wird die Gruppe nicht bereitgestellt.

Die Bereitstellungszeit hängt von der Netzwerkgeschwindigkeit ab. Bei langsamen Netzwerkgeschwindigkeiten können Sie diesen Wert erhöhen.

Standardwert: 180

Optionsname für den Hintergrundmodus: `--deployment-timeout`

Protokollpfad

Der Speicherort der Protokolldatei, die Informationen zur Greengrass-Gruppe und Core-Setup-Operationen enthält. Verwenden Sie dieses Protokoll, um Fehler bei der Bereitstellung und andere Probleme mit der Greengrass-Gruppe und dem Core-Setup zu beheben.

Standardwert: `./`

Optionsname für den Hintergrundmodus: `--log-path`

Verbosity (Ausführlichkeit)

Gibt an, ob detaillierte Protokollinformationen im Terminal gedruckt werden sollen, während das Skript ausgeführt wird. Sie können diese Informationen verwenden, um Probleme bei der Geräteeinrichtung zu beheben.

Standardwert: no

Optionsname für den Hintergrundmodus: `--verbose` – Diese Option akzeptiert keinen Wert. Fügen Sie sie in Ihren Befehl ein, wenn Sie detaillierte Protokollinformationen drucken möchten.

Ausführen von Greengrass Device Setup im Hintergrundmodus

Sie können Greengrass Device Setup im Hintergrundmodus ausführen, damit das Skript Sie nicht zur Eingabe von Werten auffordert. Wenn Sie die Ausführung im Hintergrund festlegen möchten, geben

Sie den Modus `bootstrap-greengrass` und Ihre [Eingabewerte](#) nach dem Start des Skripts an. Sie können Eingabewerte weglassen, wenn Sie die Standardwerte verwenden möchten.

Das Verfahren hängt davon ab, ob Sie Ihre AWS-Konto Anmeldeinformationen als Umgebungsvariablen angeben, bevor Sie das Skript starten, oder als Eingabewerte, nachdem Sie das Skript gestartet haben.

Bereitstellen der Anmeldeinformationen als Umgebungsvariablen

1. [Geben Sie Ihre Anmeldeinformationen](#) als Umgebungsvariablen an. Im folgenden Beispiel werden temporäre Anmeldeinformationen exportiert, die das Sitzungstoken enthalten.

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

Note

Wenn Sie das Greengrass-Geräte-Setup auf einem Raspbian oder einer OpenWrt Plattform ausführen, erstellen Sie eine Kopie dieser Befehle. Sie müssen sie nach dem Neustart des Geräts erneut angeben.

2. Laden Sie das Skript herunter und starten Sie es. Geben Sie die Eingabewerte wie erforderlich an. Zum Beispiel:

- Wenn Sie alle Standardwerte verwenden möchten:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
```

- Wenn Sie benutzerdefinierte Werte angeben möchten:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--region us-east-1
--group-name Custom_Group_Name
--core-name Custom_Core_Name
```

```
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

 Note

Wenn Sie das Skript unter Verwendung von `curl` herunterladen möchten, ersetzen Sie `wget -q -O` im Befehl durch `curl`.

3. Wenn auf Ihrem Kerngerät Raspbian oder ausgeführt wird OpenWrt, starten Sie das Gerät neu, wenn Sie dazu aufgefordert werden, geben Sie Ihre Anmeldeinformationen ein und starten Sie dann das Skript neu.
 - a. Wenn Sie zum Neustart des Geräts aufgefordert werden, führen Sie einen der folgenden Befehle aus.

Für Raspbian Plattformen:

```
sudo reboot
```

Für OpenWrt Plattformen:

```
reboot
```

- b. Öffnen Sie nach dem Neustart des Geräts das Terminal und geben Sie Ihre Anmeldeinformationen als Umgebungsvariablen an.

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

- c. Starten Sie das Skript erneut.

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
```

- d. Wenn Sie aufgefordert werden, die Eingabewerte aus der vorherigen Sitzung zu verwenden oder eine neue Installation zu starten, geben Sie `yes` ein, um Ihre Eingabewerte wiederzuverwenden.

 Note

Auf Plattformen, für die ein Neustart erforderlich ist, werden Ihre Eingabewerte (mit Ausnahme der Anmeldeinformationen) der vorherigen Sitzung vorübergehend in der Datei `GreengrassDeviceSetup.config.info` gespeichert.

Wenn die Einrichtung abgeschlossen ist, zeigt das Terminal eine Erfolgsstatusmeldung an, die in etwa wie folgt aussieht.

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mu1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

=====
```

4. Wenn Sie die Hello World Lambda-Funktion hinzugefügt haben, stellt das Greengrass-Geräte-Setup die Greengrass-Gruppe auf Ihrem Kerngerät bereit. Um die Lambda-Funktion zu testen oder Informationen darüber zu erhalten, wie Sie die Lambda-Funktion aus der Gruppe entfernen können, fahren Sie mit Modul 3-1 des Tutorials Erste Schritte fort. [the section called “Stellen Sie sicher, dass die Lambda-Funktion auf dem Kerngerät ausgeführt wird”](#)

 Note

Vergewissern Sie sich, dass es sich bei der AWS-Region Auswahl in der Konsole um dieselbe handelt, die Sie zur Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

Wenn Sie die Hello World Lambda-Funktion nicht enthalten haben, können Sie [Ihre eigene Lambda-Funktion erstellen](#) oder andere Greengrass-Funktionen ausprobieren. Beispielsweise können Sie den [Docker-Anwendungsbereitstellungs](#)-Konnektor zu Ihrer Gruppe hinzufügen und verwenden, um Docker-Container auf Ihrem Core-Gerät bereitzustellen.

Bereitstellen der Anmeldeinformationen als Eingabewerte

1. Laden Sie das Skript herunter und starten Sie es. [Geben Sie Ihre Anmeldeinformationen](#) und alle anderen Eingabewerte an, die Sie festlegen möchten. Die folgenden Beispiele zeigen, wie temporäre Anmeldeinformationen bereitgestellt werden, die das Sitzungstoken enthalten.

- Wenn Sie alle Standardwerte verwenden möchten:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- Wenn Sie benutzerdefinierte Werte angeben möchten:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

```
--region us-east-1
--group-name Custom_Group_Name
--core-name Custom_Core_Name
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

Note

Wenn Sie das Greengrass-Geräte-Setup auf einem Raspbian oder einer OpenWrt Plattform ausführen, erstellen Sie eine Kopie Ihrer Anmeldeinformationen. Sie müssen sie nach dem Neustart des Geräts erneut angeben.

Wenn Sie das Skript unter Verwendung von `curl` herunterladen möchten, ersetzen Sie `wget -q -O` im Befehl durch `curl`.

2. Wenn auf Ihrem Kerngerät Raspbian oder ausgeführt wird OpenWrt, starten Sie das Gerät neu, wenn Sie dazu aufgefordert werden, geben Sie Ihre Anmeldeinformationen ein und starten Sie dann das Skript neu.
 - a. Wenn Sie zum Neustart des Geräts aufgefordert werden, führen Sie einen der folgenden Befehle aus.

Für Raspbian Plattformen:

```
sudo reboot
```

Für OpenWrt Plattformen:

```
reboot
```

- b. Starten Sie das Skript erneut. Sie müssen Ihre Anmeldeinformationen in den Befehl einfügen, die anderen Eingabewerte jedoch nicht. Zum Beispiel:

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
--aws-access-key-id AKIAIOSFODNN7EXAMPLE
--aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

```
--aws-session-token AQoDYXdzEJr1K...o50ytwEXAMPLE=
```

- c. Wenn Sie aufgefordert werden, die Eingabewerte aus der vorherigen Sitzung zu verwenden oder eine neue Installation zu starten, geben Sie `yes` ein, um Ihre Eingabewerte wiederzuverwenden.

 Note

Auf Plattformen, für die ein Neustart erforderlich ist, werden Ihre Eingabewerte (mit Ausnahme der Anmeldeinformationen) der vorherigen Sitzung vorübergehend in der Datei `GreengrassDeviceSetup.config.info` gespeichert.

Wenn die Einrichtung abgeschlossen ist, zeigt das Terminal eine Erfolgsstatusmeldung an, die in etwa wie folgt aussieht.

```
=====
Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.

Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/vers
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mu1lv
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.
=====
```

3. Wenn Sie die Hello World Lambda-Funktion hinzugefügt haben, stellt das Greengrass-Geräte-Setup die Greengrass-Gruppe auf Ihrem Kerngerät bereit. Um die Lambda-Funktion zu testen oder Informationen darüber zu erhalten, wie Sie die Lambda-Funktion aus der Gruppe entfernen können, fahren Sie mit Modul 3-1 des Tutorials Erste Schritte fort. [the section called “Stellen Sie sicher, dass die Lambda-Funktion auf dem Kerngerät ausgeführt wird”](#)

 Note

Vergewissern Sie sich, dass es sich bei der AWS-Region Auswahl in der Konsole um dieselbe handelt, die Sie zur Konfiguration Ihrer Greengrass-Umgebung verwendet haben. Standardmäßig ist die Region USA West (Oregon).

Wenn Sie die Hello World Lambda-Funktion nicht enthalten haben, können Sie [Ihre eigene Lambda-Funktion erstellen](#) oder andere Greengrass-Funktionen ausprobieren. Beispielsweise können Sie den [Docker-Anwendungsbereitstellungs](#)-Konnektor zu Ihrer Gruppe hinzufügen und verwenden, um Docker-Container auf Ihrem Core-Gerät bereitzustellen.

Modul 1: Umgebungseinrichtung für Greengrass

Dieses Modul zeigt Ihnen, wie Sie einen out-of-the-box Raspberry Pi, eine EC2 Amazon-Instance oder ein anderes Gerät so einrichten, dass es AWS IoT Greengrass als Ihr AWS IoT Greengrass Kerngerät verwendet werden kann.

 Tip

Wie Sie vorgehen können, um das Core-Gerät durch ein Skript einrichten zu lassen, erfahren Sie unter [the section called “Schnellstart: Greengrass-Geräteeinrichtung”](#).

Dieses Modul sollte nicht mehr als 30 Minuten in Anspruch nehmen.

Bevor Sie beginnen, lesen Sie die [Anforderungen](#) für dieses Tutorial. Befolgen Sie dann die Installationsanweisungen in einem der folgenden Themen. Wählen Sie nur das Thema aus, das für Ihren Core-Gerätetyp gilt.

Themen

- [Einrichten eines Raspberry Pi](#)
- [Eine EC2 Amazon-Instance einrichten](#)
- [Einrichten anderer Geräte](#)

Note

Informationen zur Verwendung von AWS IoT Greengrass Running in einem vorgefertigten Docker-Container finden Sie unter [the section called “ AWS IoT Greengrass In einem Docker-Container ausführen”](#)

Einrichten eines Raspberry Pi

Folgen Sie den Schritten in diesem Thema, um einen Raspberry Pi für die Verwendung als AWS IoT Greengrass Core einzurichten.

Tip

AWS IoT Greengrass bietet auch andere Optionen für die Installation der AWS IoT Greengrass Core-Software. Sie können beispielsweise das [Greengrass-Geräte-Setup](#) verwenden, um Ihre Umgebung zu konfigurieren und die neueste Version der AWS IoT Greengrass Core-Software zu installieren. Oder Sie können auf unterstützten Debian-Plattformen den [APT-Paketmanager](#) verwenden, um die AWS IoT Greengrass Core-Software zu installieren oder zu aktualisieren. Weitere Informationen finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

Wenn Sie zum ersten Mal einen Raspberry Pi einrichten, befolgen Sie alle nachfolgenden Schritte. Andernfalls können Sie mit [Schritt 9](#) fortfahren. Wir empfehlen ein Reimaging des Raspberry Pi mit dem Betriebssystem, wie in Schritt 2 empfohlen.

1. Laden Sie einen SD-Kartenformatierer wie den [SD Memory Card](#) Formatter herunter und installieren Sie ihn. Legen Sie die SD-Karte in Ihren Computer. Starten Sie das Programm und wählen Sie das Laufwerk aus, in das Sie die SD-Karte eingelegt haben. Sie können eine Schnellformatierung der SD-Karte durchführen.
2. Laden Sie das Betriebssystem [Raspbian Buster](#) als zip-Datei herunter.
3. Folgen Sie in einem SD-Karten-Schreibtool (z. B. [Etcher](#)) den Anweisungen im Tool, um die heruntergeladene Datei zip auf die SD-Karte zu flashen. Da das Betriebssystem-Image sehr groß ist, kann dieser Schritt einige Zeit in Anspruch nehmen. Werfen Sie die SD-Karte aus und setzen Sie die microSD-Karte in Ihren Raspberry Pi ein.

4. Für den ersten Systemstart empfehlen wir, dass Sie den Raspberry Pi (über HDMI) mit einem Monitor, einer Tastatur und einer Maus verbinden. Schließen Sie den Pi an eine Micro USB-Stromquelle an. Das Raspbian-Betriebssystem sollte gestartet werden.
5. Sie möchten vielleicht zuerst das Pi-Tastaturlayout konfigurieren, bevor Sie fortfahren. Wählen Sie dazu das Raspberry-Symbol in der rechten oberen Ecke, Preferences (Präferenzen) und danach Mouse and Keyboard Settings (Maus- und Tastatureinstellungen) aus. Wählen Sie als Nächstes auf der Registerkarte Keyboard (Tastatur) die Option Keyboard Layout (Tastaturlayout) und die entsprechende Tastaturvariante aus.
6. [Verbinden Sie Ihren Raspberry Pi mit dem Internet über ein WLAN-Netzwerk](#) oder ein Ethernet-Kabel.

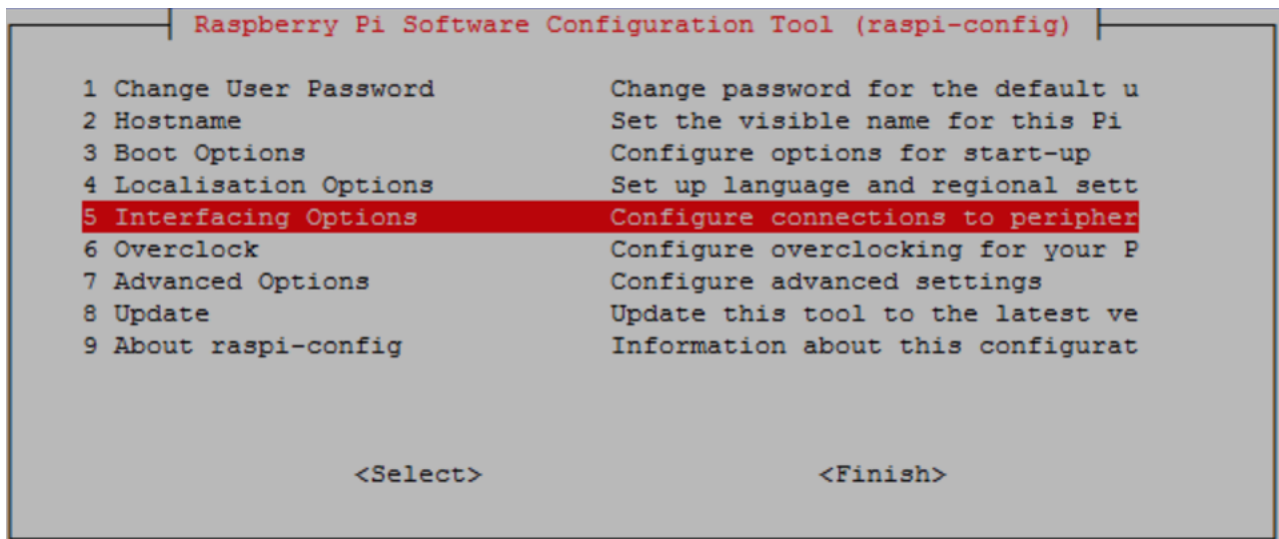
 Note

Verbinden Sie Ihren Raspberry Pi mit demselben Netzwerk wie Ihren Computer und stellen Sie sicher, dass sowohl der Computer als auch der Raspberry Pi Internetzugriff haben, bevor Sie fortfahren. Wenn Sie sich in einer Arbeitsumgebung oder hinter einer Firewall befinden, müssen Sie möglicherweise Ihren Pi und Ihren Computer mit dem Gastnetzwerk verbinden, sodass sich beide Geräte in demselben Netzwerk befinden. Bei diesem Ansatz wird jedoch möglicherweise die Verbindung Ihres Computer mit lokalen Netzwerkressourcen wie etwa Ihrem Intranet getrennt. Eine Lösung ist, den Pi mit dem WLAN-Gastnetzwerk zu verbinden und Ihren Computer über ein Ethernet-Kabel mit dem WLAN-Gastnetzwerk und Ihrem lokalen Netzwerk zu verbinden. In dieser Konfiguration sollte Ihr Computer in der Lage sein, über das WLAN-Gastnetzwerk eine Verbindung zum Raspberry Pi und über das Ethernet-Kabel eine Verbindung zu Ihren lokalen Netzwerkressourcen herzustellen.

7. Sie müssen [SSH](#) auf dem Pi einrichten, um eine Remoteverbindung zu ermöglichen. Öffnen Sie auf dem Raspberry Pi ein [Terminalfenster](#) und führen Sie den folgenden Befehl aus:

```
sudo raspi-config
```

Sie sollten Folgendes sehen:



Führen Sie einen Bildlauf nach unten aus und wählen Sie Interfacing Options und P2 SSH aus. Wählen Sie nach Aufforderung die Option Yes (Ja) aus. (Verwenden Sie die Tab-Taste gefolgt von Enter). SSH sollte damit aktiviert sein. Wählen Sie OK aus. Wählen Sie mit der Tab-Taste Finish (Fertig stellen) und drücken Sie dann Enter. Wenn der Raspberry Pi nicht automatisch neu startet, führen Sie den folgenden Befehl aus:

```
sudo reboot
```

8. Führen Sie auf dem Raspberry Pi in dem Terminal den folgenden Befehl aus:

```
hostname -I
```

Damit wird die IP-Adresse Ihres Raspberry Pi zurückgegeben.

Note

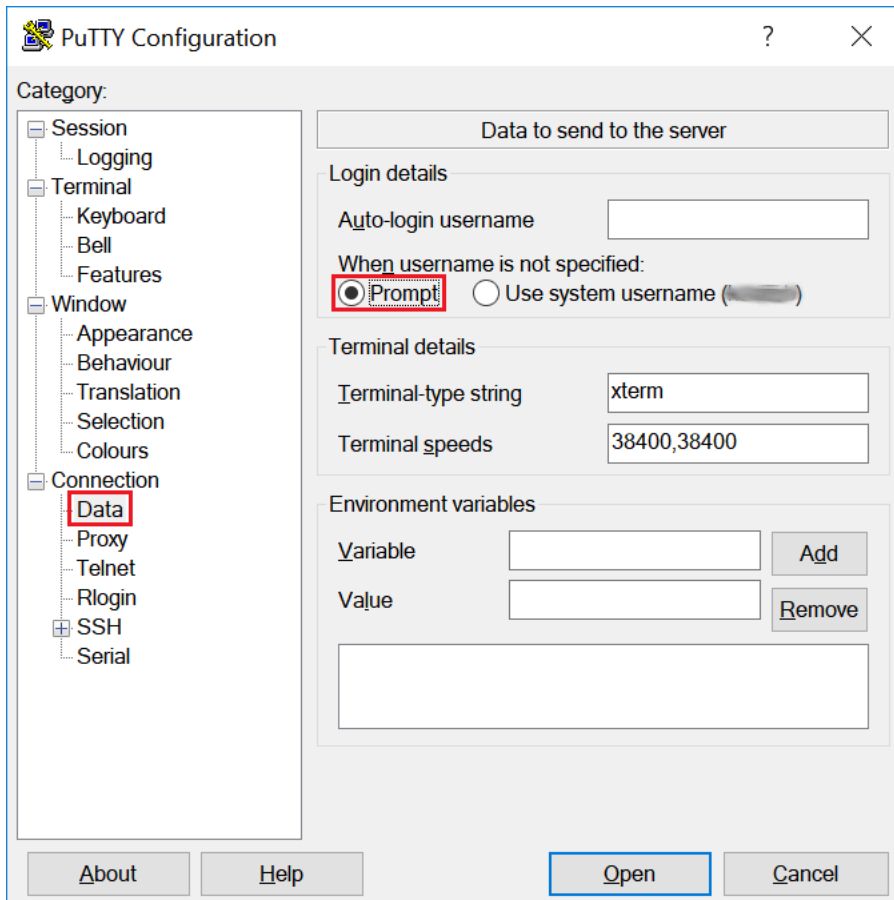
Wenn Sie nachfolgend eine Meldung hinsichtlich des Fingerabdrucks für den ECDSA-Schlüssel (Are you sure you want to continue connecting (yes/no)?) erhalten, geben Sie yes ein. Das Standardpasswort für den Raspberry Pi lautet **raspberry**.

Wenn Sie macOS verwenden, öffnen Sie ein Terminalfenster und geben Sie Folgendes ein:

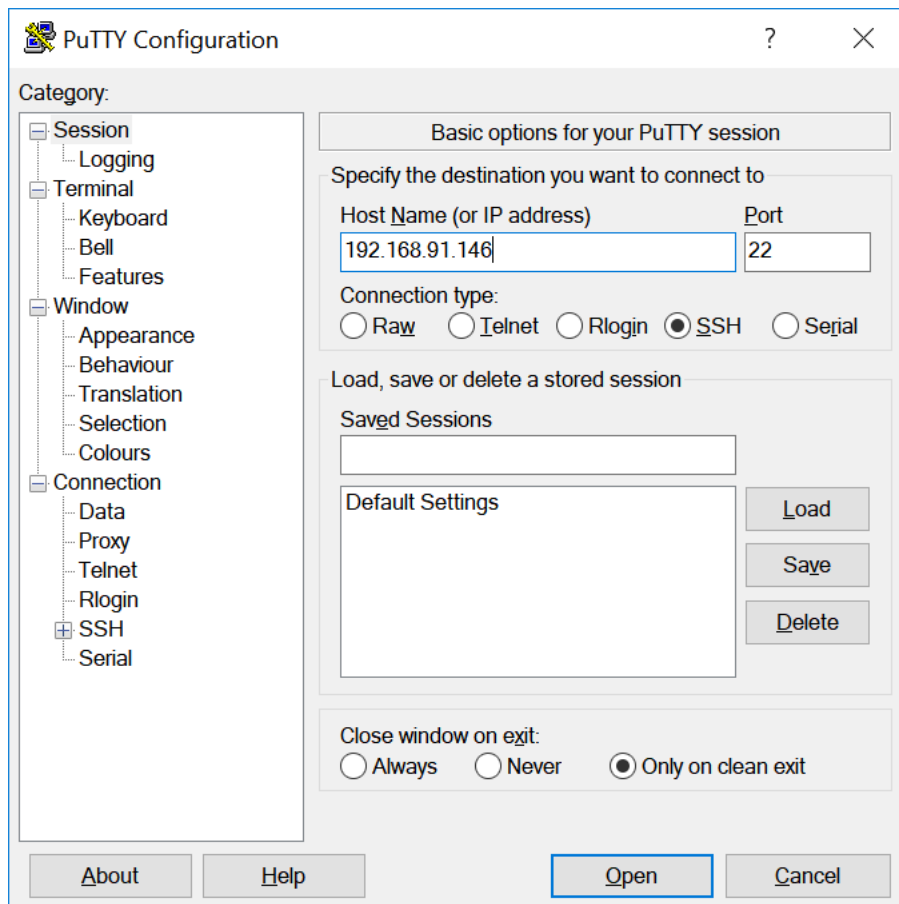
```
ssh pi@IP-address
```

IP-address ist die IP-Adresse Ihres Raspberry Pi, die Sie mit dem `hostname -I` Befehl erhalten haben.

Wenn Sie Windows verwenden, müssen Sie [PuTTY](#) installieren und konfigurieren. Erweitern Sie Connection (Verbindung), wählen Sie Data (Daten) und vergewissern Sie sich, dass Prompt (Auffordern) ausgewählt ist:

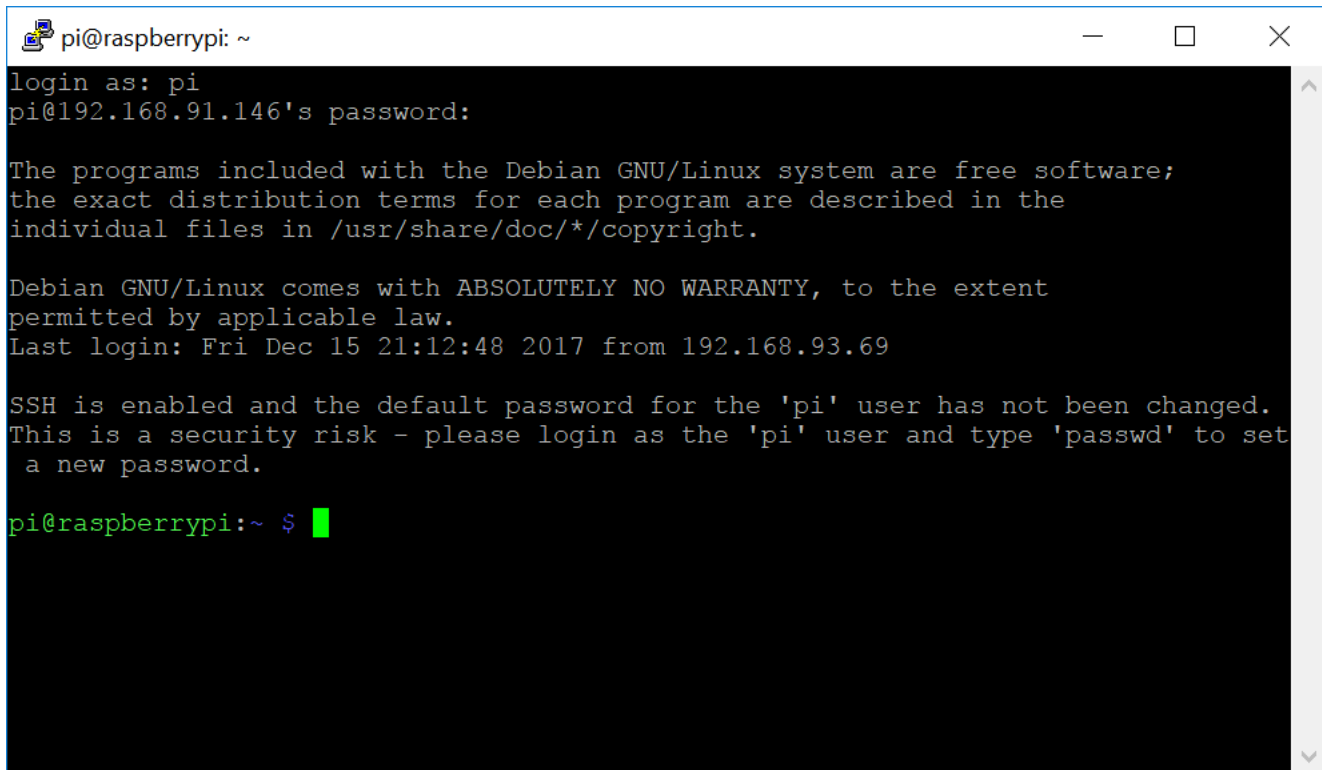


Wählen Sie Session (Sitzung), geben Sie die IP-Adresse des Raspberry Pi ein und wählen Sie dann Open (Öffnen) unter Verwendung der Standardeinstellungen.



Wenn eine PuTTY-Sicherheitswarnung angezeigt wird, klicken Sie auf Yes (Ja).

Die Standardanmeldedaten und das Passwort für den Raspberry Pi lauten **pi** und **raspberry**.



```
pi@raspberrypi: ~
login as: pi
pi@192.168.91.146's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Dec 15 21:12:48 2017 from 192.168.93.69

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $
```

Note

Wenn Ihr Computer mit einem Remote-Netzwerk über VPN verbunden ist, haben Sie möglicherweise Probleme, mit SSH eine Verbindung des Computers mit dem Raspberry Pi herzustellen.

9. Sie sind jetzt bereit, den Raspberry Pi für einzurichten AWS IoT Greengrass. Führen Sie zuerst die folgenden Befehle in einem lokalen Raspberry Pi-Terminalfenster oder einem SSH-Terminalfenster aus:

Tip

AWS IoT Greengrass bietet auch andere Optionen für die Installation der AWS IoT Greengrass Core-Software. Sie können beispielsweise das [Greengrass-Geräte-Setup](#) verwenden, um Ihre Umgebung zu konfigurieren und die neueste Version der AWS IoT Greengrass Core-Software zu installieren. Oder Sie können auf unterstützten Debian-Plattformen den [APT-Paketmanager](#) verwenden, um die AWS IoT Greengrass Core-Software zu installieren oder zu aktualisieren. Weitere Informationen finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

10. Um die Sicherheit auf dem Pi-Gerät zu verbessern, aktivieren Sie den Hard- und Softlink(Symlink) -Schutz beim Betriebssystemstart.

- a. Navigieren Sie zur Datei `98-rpi.conf`.

```
cd /etc/sysctl.d
ls
```

 Note

Wenn die Datei `98-rpi.conf` nicht angezeigt wird, befolgen Sie die Anweisungen in der Datei `README.sysctl`.

- b. Fügen Sie die beiden folgenden Zeilen mithilfe eines Texteditors (z. B. Leafpad, GNU nano oder vi) am Ende der Datei hinzu. Sie müssen für die Bearbeitung als Root-Benutzer möglicherweise den Befehl `sudo` verwenden (z. B. `sudo nano 98-rpi.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

- c. Starten Sie den Pi neu.

```
sudo reboot
```

Stellen Sie nach ca. einer Minute unter Verwendung von SSH eine Verbindung mit dem Pi her und führen Sie dann den folgenden Befehl aus, um die Änderung zu bestätigen:

```
sudo sysctl -a 2> /dev/null | grep fs.protected
```

Sie sollten `fs.protected_hardlinks = 1` und `fs.protected_symlinks = 1` sehen.

11. Bearbeiten Sie Ihre Befehlszeilen-Startdatei, um Speicher-cgroups zu aktivieren und zu mounten. Dadurch kann AWS IoT Greengrass das Speicherlimit für Lambda-

Funktionen festgelegt werden. Cgroups müssen auch AWS IoT Greengrass im [Standard-Containerisierungsmodus](#) ausgeführt werden.

- a. Navigieren Sie zum Verzeichnis boot.

```
cd /boot/
```

- b. Öffnen Sie mit einem Texteditor `cmdline.txt`. Fügen Sie Folgendes am Ende der vorhandenen Linie – nicht als eine neue Zeile – ein. Sie müssen für die Bearbeitung als Root-Benutzer möglicherweise den Befehl `sudo` verwenden (z. B. `sudo nano cmdline.txt`).

```
cgroup_enable=memory cgroup_memory=1
```

- c. Starten Sie den Pi neu.

```
sudo reboot
```

Ihr Raspberry Pi sollte jetzt für AWS IoT Greengrass bereit sein.

12. Optional. Installieren Sie die Java 8-Laufzeitumgebung, die vom [Stream-Manager](#) benötigt wird. In diesem Tutorial wird Stream-Manager nicht verwendet, jedoch wird der Workflow zur Erstellung von Standardgruppen verwendet, der Stream-Manager standardmäßig aktiviert. Verwenden Sie die folgenden Befehle, um die Java 8-Laufzeitumgebung auf dem Core-Gerät zu installieren oder den Stream-Manager zu deaktivieren, bevor Sie Ihre Gruppe bereitstellen. Anweisungen zum Deaktivieren des Stream-Managers finden Sie in Modul 3.

```
sudo apt install openjdk-8-jdk
```

13. Um sicherzustellen, dass Sie über alle erforderlichen Abhängigkeiten verfügen, laden Sie den Greengrass-Abhängigkeitsprüfer aus dem Samples-Repository herunter und führen Sie [AWS IoT Greengrass ihn](#) aus. GitHub Mit diesen Befehlen wird das Dependency Checker-Skript im Verzeichnis entpackt und ausgeführt. Downloads

Note

Die Abhängigkeitsprüfung schlägt möglicherweise fehl, wenn Sie Version 5.4.51 des Raspbian-Kernels ausführen. Diese Version mountet Speicher-Cgroups nicht korrekt.

Dies kann dazu führen, dass Lambda-Funktionen, die im Container-Modus ausgeführt werden, fehlschlagen.

Weitere Informationen zur Aktualisierung Ihres Kernels finden Sie in den Raspberry Pi-Foren unter [Cgroups not loaded after kernel upgrade](#).

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

Wenn `more` erscheint, drücken Sie die Taste Spacebar, um einen weiteren Bildschirm mit Text anzuzeigen.

Important

Dieses Tutorial benötigt die Python 3.7-Laufzeit, um lokale Lambda-Funktionen auszuführen. Wenn der Stream-Manager aktiviert ist, ist auch die Java 8-Laufzeitumgebung erforderlich. Wenn das Skript `check_ggc_dependencies` Warnmeldungen zu diesen fehlenden Laufzeitvoraussetzungen ausgibt, installieren Sie sie, bevor Sie fortfahren. Sie können Warnmeldungen zu anderen fehlenden optionalen Laufzeitvoraussetzungen ignorieren.

Weitere Information zum Befehl `modprobe` erhalten Sie, wenn Sie `man modprobe` im Terminal ausführen.

Die Konfiguration Ihres Raspberry Pi ist damit abgeschlossen. Fahren Sie fort mit [the section called “Modul 2: Installation der AWS IoT Greengrass Core-Software”](#).

Eine EC2 Amazon-Instance einrichten

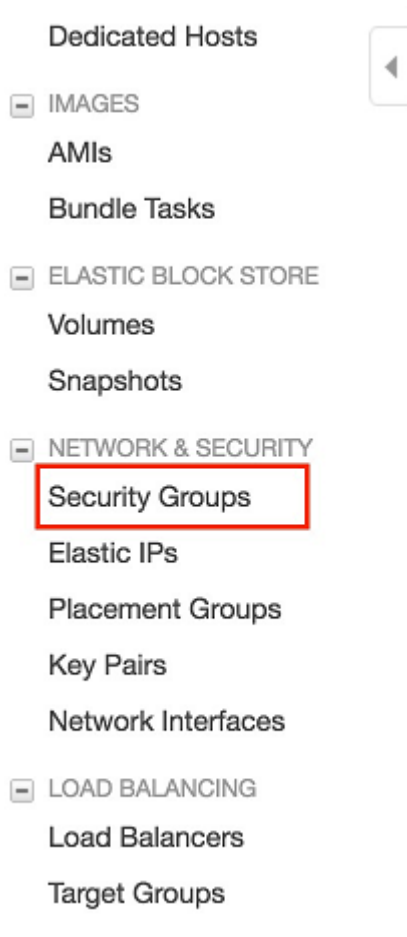
Folgen Sie den Schritten in diesem Thema, um eine EC2 Amazon-Instance einzurichten, die Sie als Ihren AWS IoT Greengrass Kern verwenden möchten.

Tip

Oder, wie Sie ein Skript verwenden, das Ihre Umgebung einrichtet und die AWS IoT Greengrass Core-Software für Sie installiert, finden Sie unter [the section called “Schnellstart: Greengrass-Geräteeinrichtung”](#).

Sie können dieses Tutorial zwar mit einer EC2 Amazon-Instance abschließen, AWS IoT Greengrass sollte aber idealerweise mit physischer Hardware verwendet werden. Wir empfehlen, dass Sie nach Möglichkeit [einen Raspberry Pi einrichten](#), anstatt eine EC2 Amazon-Instance zu verwenden. Wenn Sie einen Raspberry Pi verwenden, müssen Sie die Schritte in diesem Thema nicht ausführen.

1. Melden Sie sich bei der an [AWS-Managementkonsole](#) und starten Sie eine EC2 Amazon-Instance mit einem Amazon Linux AMI. Informationen zu EC2 Amazon-Instances finden Sie im [Amazon EC2 Getting Started Guide](#).
2. Nachdem Ihre EC2 Amazon-Instance läuft, aktivieren Sie Port 8883, um eingehende MQTT-Kommunikation zuzulassen, sodass sich andere Geräte mit dem AWS IoT Greengrass Core verbinden können.
 - a. Wählen Sie im Navigationsbereich der EC2 Amazon-Konsole die Option Sicherheitsgruppen aus.



- b. Wählen Sie die Sicherheitsgruppe für die Instance aus, die Sie gerade gestartet haben, und klicken Sie dann auf die Registerkarte Regeln für eingehenden Datenverkehr.
- c. Wählen Sie Edit inbound rules (Regeln für eingehenden Datenverkehr bearbeiten) aus.

Um Port 8883 zu aktivieren, fügen Sie eine benutzerdefinierte TCP-Regel zur Sicherheitsgruppe hinzu. Weitere Informationen finden Sie unter [Regeln zu einer Sicherheitsgruppe hinzufügen](#) im EC2 Amazon-Benutzerhandbuch.

- d. Wählen Sie auf der Seite Regeln für eingehenden Datenverkehr bearbeiten die Option Regel hinzufügen aus, geben Sie die folgenden Einstellungen ein und wählen Sie dann Speichern.
 - Wählen Sie für Type Custom TCP Rule aus.
 - Geben **8883** Sie für Portbereich den Wert ein.
 - Wählen Sie unter Source (Quelle) die Option Anywhere (Alle) aus.
 - Geben Sie für Beschreibung den Text **MQTT Communications** ein.

3. Connect zu Ihrer EC2 Amazon-Instance her.
 - a. Wählen Sie im Navigationsbereich Instances, wählen Sie Ihre Instance aus und klicken Sie dann auf Connect (Verbinden).
 - b. Befolgen Sie die Anweisungen auf der Seite Connect To Your Instance (Mit Ihrer Instance verbinden), um eine Verbindung mit Ihrer Instance [über SSH](#) und Ihre private Schlüsseldatei herzustellen.

Sie können [PuTTY](#) für Windows oder Terminal für macOS verwenden. Weitere Informationen finden Sie unter [Connect to your Linux Instance](#) im EC2 Amazon-Benutzerhandbuch.

Sie sind jetzt bereit, Ihre EC2 Amazon-Instance für einzurichten AWS IoT Greengrass.

4. Nachdem Sie mit Ihrer EC2 Amazon-Instance verbunden sind, erstellen Sie die ggc_group Konten ggc_user und:

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

 Note

Wenn der Befehl `adduser` in Ihrem System nicht verfügbar ist, verwenden Sie den folgenden Befehl.

```
sudo useradd --system ggc_user
```

5. Um die Sicherheit zu verbessern, stellen Sie sicher, dass der Hardlink- und Softlink-Schutz (Symlink) auf dem Betriebssystem der EC2 Amazon-Instance beim Start aktiviert ist.

 Note

Die Schritte für die Aktivierung des Hardlink- und Softlink-Schutzes sind vom Betriebssystem abhängig. Weitere Informationen finden Sie in der Dokumentation für Ihre Verteilung.

- a. Führen Sie den folgenden Befehl aus, um zu prüfen, ob der Hardlink- und Softlink-Schutz aktiviert ist:

```
sudo sysctl -a | grep fs.protected
```

Wenn Hardlinks und Softlinks auf 1 festgelegt sind, sind die Schutzfunktionen korrekt aktiviert. Fahren Sie mit Schritt 6 fort.

 Note

Softlinks werden durch `fs.protected_symlinks` dargestellt.

- b. Wenn Hardlinks und Softlinks nicht auf 1 festgelegt sind, aktivieren Sie diese Schutzfunktionen. Navigieren Sie zur Systemkonfigurationsdatei.

```
cd /etc/sysctl.d  
ls
```

- c. Fügen Sie in Ihrem bevorzugten Texteditor (z. B. Leafpad, GNU nano oder vi) am Ende der Systemkonfigurationsdatei die folgenden beiden Zeilen hinzu. In Amazon Linux 1 ist dies die Datei `00-defaults.conf`. In Amazon Linux 2 ist dies die Datei `99-amazon.conf`. Möglicherweise müssen Sie die Berechtigungen ändern (mit dem Befehl `chmod`), damit Sie in die Datei schreiben können. Sie können auch den Befehl `sudo nano 00-defaults.conf` verwenden, um die Bearbeitung als Root-Benutzer auszuführen (z. B. `sudo`).

```
fs.protected_hardlinks = 1  
fs.protected_symlinks = 1
```

- d. Starten Sie die EC2 Amazon-Instance neu.

```
sudo reboot
```

Stellen Sie nach ein paar Minuten eine Verbindung mit Ihrer Instance über SSH her und führen Sie dann den folgenden Befehl aus, um die Änderung zu bestätigen.

```
sudo sysctl -a | grep fs.protected
```

Die Hard- und Softlinks sind auf 1 eingestellt.

6. Extrahieren Sie das folgende Skript und führen Sie es aus, um [Linux-Kontrollgruppen](#) (Cgroups) zu mounten. Dadurch kann AWS IoT Greengrass das Speicherlimit für Lambda-Funktionen festgelegt werden. Cgroups müssen auch AWS IoT Greengrass im [Standard-Containerisierungsmodus](#) ausgeführt werden.

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

Ihre EC2 Amazon-Instance sollte jetzt bereit sein für AWS IoT Greengrass.

7. Optional. Installieren Sie die Java 8-Laufzeitumgebung, die vom [Stream-Manager](#) benötigt wird. In diesem Tutorial wird Stream-Manager nicht verwendet, jedoch wird der Workflow zur Erstellung von Standardgruppen verwendet, der Stream-Manager standardmäßig aktiviert. Verwenden Sie die folgenden Befehle, um die Java 8-Laufzeitumgebung auf dem Core-Gerät zu installieren oder den Stream-Manager zu deaktivieren, bevor Sie Ihre Gruppe bereitstellen. Anweisungen zum Deaktivieren des Stream-Managers finden Sie in Modul 3.

- Für Debian-basierte Distributionen:

```
sudo apt install openjdk-8-jdk
```

- Für Red Hat-basierte Distributionen:

```
sudo yum install java-1.8.0-openjdk
```

8. Um sicherzustellen, dass Sie über alle erforderlichen Abhängigkeiten verfügen, laden Sie den Greengrass-Abhängigkeitsprüfer aus dem Samples-Repository herunter und führen Sie [AWS IoT Greengrass ihn](#) aus. GitHub Mit diesen Befehlen wird das Dependency Checker-Skript in Ihrer EC2 Amazon-Instance heruntergeladen, entpackt und ausgeführt.

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

 Important

Dieses Tutorial benötigt die Python 3.7-Laufzeit, um lokale Lambda-Funktionen auszuführen. Wenn der Stream-Manager aktiviert ist, ist auch die Java 8-Laufzeitumgebung erforderlich. Wenn das Skript `check_ggc_dependencies` Warnmeldungen zu diesen fehlenden Laufzeitvoraussetzungen ausgibt, installieren Sie sie, bevor Sie fortfahren. Sie können Warnmeldungen zu anderen fehlenden optionalen Laufzeitvoraussetzungen ignorieren.

Ihre EC2 Amazon-Instance-Konfiguration ist abgeschlossen. Fahren Sie fort mit [the section called “Modul 2: Installation der AWS IoT Greengrass Core-Software”](#).

Einrichten anderer Geräte

Folgen Sie den Schritten in diesem Thema, um ein Gerät (außer einem Raspberry Pi) für die Verwendung als AWS IoT Greengrass Core einzurichten.

 Tip

Oder, wie Sie ein Skript verwenden, das Ihre Umgebung einrichtet und die AWS IoT Greengrass Core-Software für Sie installiert, finden Sie unter [the section called “Schnellstart: Greengrass-Geräteeinrichtung”](#).

Wenn Sie noch nicht damit vertraut sind AWS IoT Greengrass, empfehlen wir Ihnen, einen Raspberry Pi oder eine EC2 Amazon-Instance als Ihr Kerngerät zu verwenden und die für Ihr Gerät geeigneten [Einrichtungsschritte](#) zu befolgen.

Wenn du vorhast, mithilfe des Yocto-Projekts ein benutzerdefiniertes Linux-basiertes System zu erstellen, kannst du das AWS IoT Greengrass Bitbake-Rezept aus dem Projekt verwenden. `meta-aws` Dieses Rezept hilft Ihnen auch bei der Entwicklung einer Softwareplattform, die AWS Edge-Software für eingebettete Anwendungen unterstützt. Der Bitbake-Build installiert, konfiguriert und führt die AWS IoT Greengrass Core-Software automatisch auf Ihrem Gerät aus.

Yocto-Projekt

Ein Open-Source-Kooperationsprojekt, das Ihnen hilft, maßgeschneiderte Linux-basierte Systeme für eingebettete Anwendungen unabhängig von der Hardwarearchitektur zu entwickeln. Weitere Informationen finden Sie im [Yocto-Projekt](#).

meta-aws

Ein AWS verwaltetes Projekt, das Yocto-Rezepte bereitstellt. Sie können die Rezepte verwenden, um AWS Edge-Software in Linux-basierten Systemen zu entwickeln, die mit [OpenEmbedded](#) Yocto Project erstellt wurden. Weitere Informationen zu dieser von der Community unterstützten Funktion finden Sie im Projekt unter [meta-aws](#) GitHub

meta-aws-demos

Ein AWS verwaltetes Projekt, das Demonstrationen für das meta-aws Projekt enthält. Weitere Beispiele zum Integrationsprozess finden Sie im [meta-aws-demos](#) Projekt unter GitHub.

Um ein anderes Gerät oder eine andere [unterstützte Plattform](#) zu verwenden, folgen Sie den Schritten in diesem Thema.

1. Wenn es sich bei Ihrem Kerngerät um ein NVIDIA Jetson-Gerät handelt, müssen Sie zuerst die Firmware mit dem JetPack 4.3-Installationsprogramm flashen. Wenn Sie ein anderes Gerät konfigurieren, fahren Sie mit Schritt 2 fort.

Note

Die von Ihnen verwendete JetPack Installationsversion basiert auf Ihrer CUDA Toolkit-Zielversion. Die folgenden Anweisungen verwenden JetPack 4.3 und CUDA Toolkit 10.0. Informationen zur Verwendung der für Ihr Gerät geeigneten Versionen finden Sie unter [How to Install Jetpack](#) in der NVIDIA-Dokumentation.

- a. Flashen Sie auf einem physischen Desktop, auf dem Ubuntu 16.04 oder höher ausgeführt wird, die Firmware mit dem JetPack 4.3-Installationsprogramm, wie in [Herunterladen und Installieren JetPack](#) (4.3) in der NVIDIA-Dokumentation beschrieben.

Befolgen Sie die Anweisungen im Installationsprogramm zum Installieren aller Pakete und Abhängigkeiten auf der Jetson-Karte, die über ein Micro-B-Kabel mit dem Desktop verbunden sein muss.

- b. Starten Sie Ihre Karte im normalen Modus neu und schließen Sie einen Bildschirm an.

 Note

Wenn Sie mit SSH eine Verbindung mit der Jetson-Platine herstellen, verwenden Sie den Standard-Benutzernamen (**nvidia**) und das Standard-Passwort (**nvidia**).

2. Führen Sie einen der folgenden Befehle aus, um den Benutzer `ggc_user` und die Gruppe `ggc_group` zu erstellen. Die von Ihnen ausgeführten Befehle unterscheiden sich je nach der auf Ihrem Core-Gerät installierten Verteilung.

- Wenn Ihr Core-Gerät läuft OpenWrt, führen Sie die folgenden Befehle aus:

```
opkg install shadow-useradd
opkg install shadow-groupadd
useradd --system ggc_user
groupadd --system ggc_group
```

- Führen Sie andernfalls die folgenden Befehle aus:

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

 Note

Wenn der Befehl `addgroup` in Ihrem System nicht verfügbar ist, verwenden Sie den folgenden Befehl.

```
sudo groupadd --system ggc_group
```

3. Optional. Installieren Sie die Java 8-Laufzeitumgebung, die vom [Stream-Manager](#) benötigt wird. In diesem Tutorial wird Stream-Manager nicht verwendet, jedoch wird der Workflow zur Erstellung von Standardgruppen verwendet, der Stream-Manager standardmäßig aktiviert. Verwenden Sie die folgenden Befehle, um die Java 8-Laufzeitumgebung auf dem Core-Gerät zu installieren oder den Stream-Manager zu deaktivieren, bevor Sie Ihre Gruppe bereitstellen. Anweisungen zum Deaktivieren des Stream-Managers finden Sie in Modul 3.

- Für Debian- oder Ubuntu-basierte Distributionen:


```
sudo apt install openjdk-8-jdk
```

- Für Red Hat-basierte Distributionen:

```
sudo yum install java-1.8.0-openjdk
```

4. Um sicherzustellen, dass Sie über alle erforderlichen Abhängigkeiten verfügen, laden Sie den Greengrass-Abhängigkeitsprüfer aus dem Samples-Repository herunter und führen Sie [AWS IoT Greengrass ihn](#) aus. GitHub Mit diesen Befehlen wird das Dependency Checker-Skript entpackt und ausgeführt.

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

Note

Das `check_ggc_dependencies` Skript läuft auf AWS IoT Greengrass unterstützten Plattformen und erfordert spezielle Linux-Systembefehle. Weitere Informationen finden Sie in der [Readme](#)-Datei des Abhängigkeitsprüftools.

5. Installieren Sie alle erforderlichen Abhängigkeiten auf Ihrem Gerät entsprechend der Ausgabe des Abhängigkeitsprüftools. Für fehlende Abhängigkeiten auf Kernel-Ebene müssen Sie Ihren Kernel möglicherweise neu kompilieren. Zum Mounting von Linux-Steuergruppen (cgroups) können Sie das Skript [cgroupsfs-mount](#) ausführen. Dadurch kann AWS IoT Greengrass das Speicherlimit für Lambda-Funktionen festgelegt werden. Cgroups müssen auch AWS IoT Greengrass im [Standard-Containerisierungsmodus](#) ausgeführt werden.

Wenn in der Ausgabe keine Fehler angezeigt werden, AWS IoT Greengrass sollte es auf Ihrem Gerät erfolgreich ausgeführt werden können.

 Important

Dieses Tutorial benötigt die Python 3.7-Laufzeit, um lokale Lambda-Funktionen auszuführen. Wenn der Stream-Manager aktiviert ist, ist auch die Java 8-Laufzeitumgebung erforderlich. Wenn das Skript `check_ggc_dependencies` Warnmeldungen zu diesen fehlenden Laufzeitvoraussetzungen ausgibt, installieren Sie sie, bevor Sie fortfahren. Sie können Warnmeldungen zu anderen fehlenden optionalen Laufzeitvoraussetzungen ignorieren.

Eine Liste der AWS IoT Greengrass Anforderungen und Abhängigkeiten finden Sie unter [the section called “Unterstützte Plattformen und Anforderungen”](#).

Modul 2: Installation der AWS IoT Greengrass Core-Software

Dieses Modul zeigt Ihnen, wie Sie die AWS IoT Greengrass Core-Software auf dem von Ihnen ausgewählten Gerät installieren. In diesem Modul erstellen Sie zunächst eine Greengrass-Gruppe und einen Core. Anschließend laden Sie die Software herunter, konfigurieren und starten sie auf Ihrem Core-Gerät. Weitere Informationen zu AWS IoT Greengrass -Core-Softwarefunktionen finden Sie unter [the section called “Konfiguriere den AWS IoT Greengrass Kern”](#).

Bevor Sie beginnen, stellen Sie sicher, dass Sie die Einrichtungsschritte in [Modul 1](#) für das von Ihnen gewählte Gerät ausgeführt haben.

 Tip

AWS IoT Greengrass bietet auch andere Optionen für die Installation der AWS IoT Greengrass Core-Software. Sie können beispielsweise das [Greengrass-Geräte-Setup](#) verwenden, um Ihre Umgebung zu konfigurieren und die neueste Version der AWS IoT Greengrass Core-Software zu installieren. Oder Sie können auf unterstützten Debian-Plattformen den [APT-Paketmanager](#) verwenden, um die AWS IoT Greengrass Core-Software zu installieren oder zu aktualisieren. Weitere Informationen finden Sie unter [the section called “Installieren Sie die AWS IoT Greengrass Core-Software”](#).

Dieses Modul sollte nicht mehr als 30 Minuten in Anspruch nehmen.

Themen

- [Bereitstellung AWS IoT eines Objekts, das als Greengrass-Kern verwendet werden kann](#)
- [Erstellen Sie eine AWS IoT Greengrass Gruppe für den Kern](#)
- [Installation und Ausführung AWS IoT Greengrass auf dem Kerngerät](#)

Bereitstellung AWS IoT eines Objekts, das als Greengrass-Kern verwendet werden kann

Greengrass-Cores sind Geräte, auf denen die AWS IoT Greengrass Core-Software zur Verwaltung lokaler IoT-Prozesse ausgeführt wird. Um einen Greengrass-Core einzurichten, erstellen Sie ein AWS IoT Ding, das ein Gerät oder eine logische Einheit darstellt, mit der eine Verbindung hergestellt wird. AWS IoT Wenn Sie ein Gerät als Objekt registrieren AWS IoT , kann dieses Gerät ein digitales Zertifikat und Schlüssel verwenden, die ihm den Zugriff AWS IoT ermöglichen. Sie verwenden eine [AWS IoT Richtlinie](#), um dem Gerät die Kommunikation mit den AWS IoT und AWS IoT Greengrass - Diensten zu ermöglichen.

In diesem Abschnitt registrieren Sie Ihr Gerät als eine AWS IoT Sache, um es als Greengrass-Core zu verwenden.

Um etwas zu erschaffen AWS IoT

1. Navigieren Sie zur [AWS IoT -Konsole](#).
2. Erweitern Sie unter Verwalten die Option Alle Geräte und wählen Sie dann Dinge aus.
3. Wählen Sie auf der Seite Objekte die Option Objekte erstellen aus.
4. Wählen Sie auf der Seite „Dinge erstellen“ die Option „Einzelnes Ding erstellen“ und dann „Weiter“ aus.
5. Gehen Sie auf der Seite „Dingeigenschaften angeben“ wie folgt vor:
 - a. Geben Sie unter Name des Dings einen Namen ein, der für Ihr Gerät steht, z. **MyGreengrassV1Core B**.
 - b. Wählen Sie Weiter aus.
6. Wählen Sie auf der Seite Gerätezertifikat konfigurieren die Option Weiter aus.
7. Führen Sie auf der Seite Richtlinien an Zertifikat anhängen einen der folgenden Schritte aus:
 - Wählen Sie eine vorhandene Richtlinie aus, die Berechtigungen gewährt, die für Kerne erforderlich sind, und wählen Sie dann Ding erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit dem verwendet AWS Cloud.

- Erstellen Sie eine neue Richtlinie, die Kerngeräteberechtigungen gewährt, und fügen Sie sie an. Gehen Sie wie folgt vor:
 - a. Wählen Sie Richtlinie erstellen aus.

Die Seite Create policy (Richtlinie erstellen) wird in einer neuen Registerkarte geöffnet.

- b. Führen Sie auf der Seite Create policy (Richtlinie erstellen) die folgenden Schritte aus:
 - i. Geben Sie als Richtlinienname einen Namen ein, der die Richtlinie beschreibt, z. **GreengrassV1CorePolicy B**.
 - ii. Wählen Sie auf der Registerkarte Richtlinienenerklärungen unter Richtlinienokument die Option JSON aus.
 - iii. Geben Sie das folgende Richtlinienokument ein. Diese Richtlinie ermöglicht es dem Core, mit dem AWS IoT Core Dienst zu kommunizieren, mit Geräteschatten zu interagieren und mit dem AWS IoT Greengrass Dienst zu kommunizieren. Informationen dazu, wie Sie den Zugriff dieser Richtlinie je nach Anwendungsfall einschränken können, finden Sie unter [Minimale AWS IoT Richtlinie für das AWS IoT Greengrass Kerngerät](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    }
  ],
}
```

```

{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot>DeleteThingShadow"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "greengrass:*"
  ],
  "Resource": [
    "*"
  ]
}
]
}

```

- iv. Wählen Sie Create (Erstellen) aus, um die Richtlinie zu erstellen.
- c. Kehren Sie mit der geöffneten Seite Richtlinien an Zertifikat anhängen zur Browser-Registerkarte zurück. Gehen Sie wie folgt vor:
 - i. Wählen Sie in der Liste Richtlinien die Richtlinie aus, die Sie erstellt haben, z. GreengrassV1CorePolicyB.

Wenn Sie die Richtlinie nicht sehen, klicken Sie auf die Schaltfläche „Aktualisieren“.

- ii. Wählen Sie Objekt erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, mit denen der Core eine Verbindung herstellt AWS IoT.

- 8. Kehren Sie mit der geöffneten Seite Richtlinien an Zertifikat anhängen zum Browser-Tab zurück. Gehen Sie wie folgt vor:
 - a. Wählen Sie in der Liste Richtlinien die Richtlinie aus, die Sie erstellt haben, z. GreengrassV1CorePolicyB.

Wenn Sie die Richtlinie nicht sehen, klicken Sie auf die Schaltfläche „Aktualisieren“.

- b. Wählen Sie Objekt erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, mit denen der Core eine Verbindung herstellt AWS IoT.

9. Laden Sie im Modal Zertifikate und Schlüssel herunterladen die Zertifikate des Geräts herunter.

 Important

Laden Sie die Sicherheitsressourcen herunter, bevor Sie „Fertig“ wählen.

Gehen Sie wie folgt vor:

- a. Wählen Sie unter Gerätezertifikat die Option Herunterladen aus, um das Gerätezertifikat herunterzuladen.
- b. Wählen Sie unter Datei mit öffentlichem Schlüssel die Option Herunterladen aus, um den öffentlichen Schlüssel für das Zertifikat herunterzuladen.
- c. Wählen Sie unter Datei mit privatem Schlüssel die Option Herunterladen aus, um die Datei mit dem privaten Schlüssel für das Zertifikat herunterzuladen.
- d. Lesen Sie die [Serverauthentifizierung](#) im AWS IoT Entwicklerhandbuch und wählen Sie das entsprechende Root-CA-Zertifikat aus. Wir empfehlen die Verwendung von Amazon Trust Services (ATS) -Endpunkten und ATS-Root-CA-Zertifikaten. Wählen Sie unter Root-CA-Zertifikate die Option Herunterladen für ein Root-CA-Zertifikat aus.
- e. Wählen Sie Fertig aus.

Notieren Sie sich die Zertifikat-ID, die in den Dateinamen für das Gerätezertifikat und die Schlüssel häufig vorkommt. Sie benötigen sie später.

Erstellen Sie eine AWS IoT Greengrass Gruppe für den Kern

AWS IoT Greengrass Gruppen enthalten Einstellungen und andere Informationen zu ihren Komponenten, wie Client-Geräten, Lambda-Funktionen und Konnektoren. Eine Gruppe definiert die Konfiguration für einen Kern, einschließlich der Art und Weise, wie seine Komponenten miteinander interagieren können.

In diesem Abschnitt erstellen Sie eine Gruppe für Ihren Kern.

 Tip

Ein Beispiel, das die AWS IoT Greengrass API verwendet, um eine Gruppe zu erstellen und bereitzustellen, finden Sie im Repository [gg_group_setup](#) unter GitHub

Um eine Gruppe für den Kern zu erstellen

1. Navigieren Sie zur [AWS IoT -Konsole](#).
2. Erweitern Sie unter Verwalten die Option Greengrass-Geräte und wählen Sie Gruppen (V1) aus.

 Note

Wenn Sie das Greengrass-Gerätemenü nicht sehen, wechseln Sie zu einem, AWS-Region das unterstützt AWS IoT Greengrass V1. Eine vollständige Liste der unterstützten Regionen finden Sie unter [AWS IoT Greengrass V1 -Endpunkte und -Kontingente](#) in Allgemeine AWS-Referenz. Sie müssen [das AWS IoT Ding für Ihren Core in einer Region erstellen](#), in der AWS IoT Greengrass V1 es verfügbar ist.

3. Wählen Sie auf der Greengrass-Gruppenseite die Option Gruppe erstellen aus.
4. Gehen Sie auf der Gruppenseite „Greengrass erstellen“ wie folgt vor:
 - a. Geben Sie als Greengrass-Gruppenname einen Namen ein, der die Gruppe beschreibt, z. B. **MyGreengrassGroup**
 - b. Wählen Sie für Greengrass Core das, AWS IoT was Sie zuvor erstellt haben, wie **MyGreengrassV1Core**.

Die Konsole wählt automatisch das Gerätezertifikat des Dings für Sie aus.

- c. Wählen Sie Create group (Gruppe erstellen) aus.

Installation und Ausführung AWS IoT Greengrass auf dem Kerngerät

 Note

Dieses Tutorial enthält Anweisungen zum Ausführen der AWS IoT Greengrass Core-Software auf einem Raspberry Pi. Sie können jedoch jedes unterstützte Gerät verwenden.

In diesem Abschnitt konfigurieren, installieren und führen Sie die AWS IoT Greengrass Core-Software auf Ihrem Core-Gerät aus.

Zur Installation und Ausführung AWS IoT Greengrass

1. Laden Sie aus dem Abschnitt [AWS IoT Greengrass Core-Software](#) in diesem Handbuch das AWS IoT Greengrass Core-Software-Installationspaket herunter. Wählen Sie das Paket aus, das am besten zu CPU-Architektur, Verteilung und Betriebssystem Ihres Core-Geräts passt.
 - Laden Sie für Raspberry Pi das Paket für die ARMv7L-Architektur und das Linux-Betriebssystem herunter.
 - Laden Sie für eine EC2 Amazon-Instance das Paket für die x86_64-Architektur und das Linux-Betriebssystem herunter.
 - Laden Sie für NVIDIA Jetson TX2 das Paket für die Armv8 (AArch64) -Architektur und das Linux-Betriebssystem herunter.
 - Laden Sie für Intel Atom das Paket für die x86_64-Architektur und das Linux-Betriebssystem herunter.
2. In den vorherigen Schritten haben Sie fünf Dateien auf Ihren Computer heruntergeladen:
 - `greengrass-OS-architecture-1.11.6.tar.gz`— Diese komprimierte Datei enthält die AWS IoT Greengrass Core-Software, die auf dem Core-Gerät ausgeführt wird.
 - `certificateId-certificate.pem.crt`— Die Gerätezertifikatsdatei.
 - `certificateId-public.pem.key`— Die öffentliche Schlüsseldatei des Gerätezertifikats.
 - `certificateId-private.pem.key`— Die private Schlüsseldatei des Gerätezertifikats.
 - `AmazonRootCA1.pem`— Die Datei der Amazon Root Certificate Authority (CA).

In diesem Schritt übertragen Sie diese Dateien von Ihrem Computer auf Ihr Kerngerät. Gehen Sie wie folgt vor:

- a. Wenn Sie die IP-Adresse Ihres Greengrass-Core-Geräts nicht kennen, öffnen Sie ein Terminal auf dem Core-Gerät und führen Sie den folgenden Befehl aus.

 Note

Dieser Befehl gibt für einige Geräte möglicherweise nicht die richtige IP-Adresse zurück. Informationen zum Abrufen der IP-Adresse Ihres Geräts finden Sie in der Dokumentation.

```
hostname -I
```

- b. Übertragen Sie diese Dateien von Ihrem Computer auf Ihr Kerngerät. Die Schritte zur Dateiübertragung variieren je nach Betriebssystem Ihres Computers. Wählen Sie Ihr Betriebssystem für Schritte, die zeigen, wie Sie Dateien an Ihr Raspberry Pi-Gerät übertragen.

 Note

Bei einem Raspberry PI lautet der Standardbenutzername **pi** und das Passwort ist **raspberry**.

Bei einem NVIDIA Jetson TX2 lautet der Standardbenutzername **nvidia** und das Standardkennwort **nvidia**.

Windows

Verwenden Sie zum Übertragen der komprimierten Dateien von Ihrem Computer zu einem Raspberry Pi-Core-Gerät ein Tool wie [WinSCP](#) oder den [PuTTY](#)-Befehl pscp. Für die Verwendung des pscp-Befehls öffnen Sie ein Eingabeaufforderungsfenster auf Ihrem Computer und führen die folgenden Schritte aus:

```
cd path-to-downloaded-files
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

Die Versionsnummer in diesem Befehl muss mit der Version Ihres AWS IoT Greengrass Core-Softwarepakets übereinstimmen.

macOS

Öffnen Sie zum Übertragen der komprimierten Dateien von Ihrem Mac auf ein Raspberry Pi-Core-Gerät ein Terminalfenster auf Ihrem Computer und führen Sie die folgenden Befehle aus. Das *path-to-downloaded-files* ist in der Regel ~/Downloads.

 Note

Sie werden möglicherweise aufgefordert, zwei Passwörter einzugeben. In diesem Fall gilt das erste Passwort für den sudo-Befehl des Mac, und das zweite ist das Passwort für den Raspberry Pi.

```
cd path-to-downloaded-files  
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi  
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi  
scp certificateId-public.pem.key pi@IP-address:/home/pi  
scp certificateId-private.pem.key pi@IP-address:/home/pi  
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

Die Versionsnummer in diesem Befehl muss mit der Version Ihres AWS IoT Greengrass Core-Softwarepakets übereinstimmen.

UNIX-like system

Öffnen Sie zum Übertragen der komprimierten Dateien von Ihrem Computer auf ein Raspberry Pi-Core-Gerät ein Terminalfenster auf Ihrem Computer und führen Sie die folgenden Befehle aus:

```
cd path-to-downloaded-files  
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi  
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi  
scp certificateId-public.pem.key pi@IP-address:/home/pi  
scp certificateId-private.pem.key pi@IP-address:/home/pi  
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

 Note

Die Versionsnummer in diesem Befehl muss mit der Version Ihres AWS IoT Greengrass Core-Softwarepakets übereinstimmen.

Raspberry Pi web browser

Wenn Sie den Webbrowser des Raspberry Pi verwendet haben, um die komprimierten Dateien herunterzuladen, sollten sich die Dateien im ~/Downloads Ordner des Pi befinden, /home/pi/Downloads z. Andernfalls sollten sich die komprimierten Dateien im ~ Ordner des Pi befinden, /home/pi z.

- Öffnen Sie auf dem Greengrass Core-Gerät ein Terminal und navigieren Sie zu dem Ordner, der die AWS IoT Greengrass Core-Software und die Zertifikate enthält. *path-to-transferred-files* Ersetzen Sie es durch den Pfad, in den Sie die Dateien auf das Kerngerät übertragen haben. Führen Sie beispielsweise auf einem Raspberry Pi aus `cd /home/pi`.

```
cd path-to-transferred-files
```

- Entpacken Sie die AWS IoT Greengrass Core-Software auf dem Core-Gerät. Führen Sie den folgenden Befehl aus, um das Softwarearchiv zu entpacken, das Sie auf das Kerngerät übertragen haben. Dieser Befehl verwendet das `-C /` Argument, um den /greengrass Ordner im Stammordner des Kerngeräts zu erstellen.

```
sudo tar -xzf greengrass-OS-architecture-1.11.6.tar.gz -C /
```

 Note

Die Versionsnummer in diesem Befehl muss mit der Version Ihres AWS IoT Greengrass Core-Softwarepakets übereinstimmen.

5. Verschieben Sie die Zertifikate und Schlüssel in den AWS IoT Greengrass Core-Softwareordner. Führen Sie die folgenden Befehle aus, um einen Ordner für Zertifikate zu erstellen und die Zertifikate und Schlüssel in diesen Ordner zu verschieben. *path-to-transferred-files* Ersetzen Sie es durch den Pfad, in den Sie die Dateien auf dem Kerngerät übertragen haben, und *certificateId* ersetzen Sie es durch die Zertifikat-ID in den Dateinamen. Ersetzen Sie beispielsweise auf einem Raspberry Pi *path-to-transferred-files* durch **/home/pi**

```
sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/certs
sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs
sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs
sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
```

6. Die AWS IoT Greengrass Core-Software verwendet eine Konfigurationsdatei, die Parameter für die Software spezifiziert. Diese Konfigurationsdatei spezifiziert die Dateipfade für Zertifikatsdateien und die zu AWS Cloud verwendenden Endpunkte. In diesem Schritt erstellen Sie die AWS IoT Greengrass Core-Softwarekonfigurationsdatei für Ihren Core. Gehen Sie wie folgt vor:
 - a. Holen Sie sich den Amazon Resource Name (ARN) für die AWS IoT Sache Ihres Kerns. Gehen Sie wie folgt vor:
 - i. Wählen Sie in der [AWS IoT Konsole](#) unter Verwalten unter Greengrass-Geräte die Option Gruppen (V1) aus.
 - ii. Wählen Sie auf der Greengrass-Gruppenseite die Gruppe aus, die Sie zuvor erstellt haben.
 - iii. Wählen Sie unter Übersicht die Option Greengrass Core aus.
 - iv. Kopieren Sie auf der Seite mit den Kerndetails den AWS IoT Ding-ARN und speichern Sie ihn zur Verwendung in der AWS IoT Greengrass Core-Konfigurationsdatei.

- b. Ruft den AWS IoT Gerätedatenendpunkt für Sie AWS-Konto in der aktuellen Region ab. Geräte verwenden diesen Endpunkt, um eine Verbindung zu AWS externen Geräten AWS IoT herzustellen. Gehen Sie wie folgt vor:
 - i. Wählen Sie in der [AWS IoT Konsole](#) Einstellungen aus.
 - ii. Kopieren Sie unter Gerätedatenendpunkt den Endpunkt und speichern Sie ihn zur Verwendung in der AWS IoT Greengrass Core-Konfigurationsdatei.
- c. Erstellen Sie die AWS IoT Greengrass Core-Software-Konfigurationsdatei. Sie können beispielsweise den folgenden Befehl ausführen, um die Datei mit GNU Nano zu erstellen.

```
sudo nano /greengrass/config/config.json
```

Ersetzen Sie den Inhalt der Datei durch das folgende JSON-Dokument.

```
{
  "coreThing" : {
    "caPath": "AmazonRootCA1.pem",
    "certPath": "certificateId-certificate.pem.crt",
    "keyPath": "certificateId-private.pem.key",
    "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
    "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
    "ggHost": "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
  "crypto": {
    "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-private.pem.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-private.pem.key",

```

```
        "certificatePath": "file:///greengrass/certs/certificateId-  
certificate.pem.crt"  
    }  
}  
}
```

Führen Sie dann die folgenden Schritte aus:

- Wenn Sie ein anderes Amazon-Root-CA-Zertifikat als Amazon Root CA 1 heruntergeladen haben, ersetzen Sie jede Instanz von *AmazonRootCA1.pem* durch den Namen der Amazon-Root-CA-Datei.
- Ersetzen Sie jede Instanz von *certificateId* durch die Zertifikat-ID im Namen des Zertifikats und der Schlüsseldateien.
- Ersetze es *arn:aws:iot:region:account-id:thing/MyGreengrassV1Core* durch den ARN des Dings deines Kerns, den du zuvor gespeichert hast.
- Ersetze es *MyGreengrassV1core* durch den Namen des Dings deines Kerns.
- *device-data-prefix-ats.iot.region.amazonaws.com* Ersetzen Sie es durch den AWS IoT Gerätedatenendpunkt, den Sie zuvor gespeichert haben.
- Ersetzen Sie *region* durch Ihren AWS-Region.

Weitere Informationen zu den Konfigurationsoptionen, die Sie in dieser Konfigurationsdatei angeben können, finden Sie unter [AWS IoT Greengrass Kernkonfigurationsdatei](#).

7. Stellen Sie sicher, dass Ihr Core-Gerät mit dem Internet verbunden ist. Starten Sie dann AWS IoT Greengrass auf Ihrem Core-Gerät.

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

Es sollte die Meldung `Greengrass successfully started` angezeigt werden. Notieren Sie sich die PID.

Note

Informationen dazu, wie Sie Ihr Core-Gerät so einrichten, dass es AWS IoT Greengrass beim Systemstart startet, finden Sie unter [the section called “Ausführen von Greengrass bei Systemstart”](#).

Sie können den folgenden Befehl ausführen, um zu überprüfen, ob die AWS IoT Greengrass Core-Software (Greengrass-Daemon) funktioniert. Ersetzen Sie es *PID-number* durch Ihre PID:

```
ps aux | grep PID-number
```

Es sollte ein Eintrag für die PID mit einem Pfad zum laufenden Greengrass-Daemon angezeigt werden (z. B. /greengrass/ggc/packages/1.11.6/bin/daemon). Falls beim Starten Probleme auftreten AWS IoT Greengrass, finden Sie weitere Informationen unter [Fehlerbehebung](#).

Modul 3 (Teil 1): Lambda-Funktionen auf AWS IoT Greengrass

Dieses Modul zeigt Ihnen, wie Sie eine Lambda-Funktion erstellen und bereitstellen, die MQTT-Nachrichten von Ihrem AWS IoT Greengrass Kerngerät sendet. Das Modul beschreibt Lambda-Funktionskonfigurationen, Abonnements, die verwendet werden, um MQTT-Messaging zu ermöglichen, und Bereitstellungen auf einem Kerngerät.

[Modul 3 \(Teil 2\)](#) behandelt die Unterschiede zwischen On-Demand-Funktionen und langlebigen Lambda-Funktionen, die AWS IoT Greengrass auf dem Kern ausgeführt werden.

Bevor Sie beginnen, stellen Sie sicher, dass Sie [Modul 1 und Modul 2](#) abgeschlossen haben und über ein laufendes AWS IoT Greengrass Kerngerät verfügen.

Tip

Wie Sie vorgehen können, um das Core-Gerät durch ein Skript einrichten zu lassen, erfahren Sie unter [the section called “Schnellstart: Greengrass-Geräteeinrichtung”](#). Das Skript kann auch die in diesem Modul verwendete Lambda-Funktion erstellen und bereitstellen.

Dieses Modul sollte etwa 30 Minuten in Anspruch nehmen.

Themen

- [Eine Lambda-Funktion erstellen und verpacken](#)
- [Konfigurieren Sie die Lambda-Funktion für AWS IoT Greengrass](#)
- [Bereitstellen von Cloud-Konfigurationen auf einem Greengrass-Core-Gerät](#)
- [Stellen Sie sicher, dass die Lambda-Funktion auf dem Kerngerät ausgeführt wird](#)

Eine Lambda-Funktion erstellen und verpacken

Die Python-Lambda-Beispielfunktion in diesem Modul verwendet das [AWS IoT Greengrass Core SDK](#) für Python, um MQTT-Nachrichten zu veröffentlichen.

In diesem Schritt führen Sie folgende Aktionen aus:

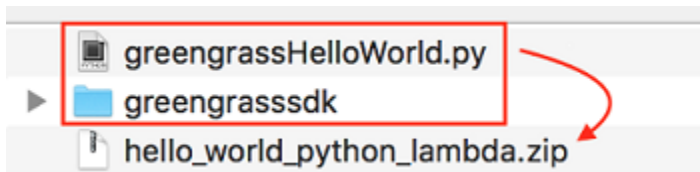
- Laden Sie das AWS IoT Greengrass Core SDK für Python auf Ihren Computer herunter (nicht auf das AWS IoT Greengrass Kerngerät).
- Erstellen Sie ein Paket zur Bereitstellung von Lambda-Funktionen, das den Funktionscode und die Abhängigkeiten enthält.
- Verwenden Sie die Lambda-Konsole, um eine Lambda-Funktion zu erstellen und das Bereitstellungspaket hochzuladen.
- Veröffentlichen Sie eine Version der Lambda-Funktion und erstellen Sie einen Alias, der auf die Version verweist.

Um dieses Modul abschließen zu können, muss Python 3.7 auf Ihrem Core-Gerät installiert sein.

1. Laden Sie von der [AWS IoT Greengrass Core SDK-Downloadseite](#) das AWS IoT Greengrass Core SDK für Python auf Ihren Computer herunter.
2. Entpacken Sie das heruntergeladene Paket, um den Lambda-Funktionscode und das SDK zu erhalten.

Die Lambda-Funktion in diesem Modul verwendet:

- Die Datei `greengrassHelloWorld.py` in `examples\HelloWorld`. Dies ist Ihr Lambda-Funktionscode. Alle fünf Sekunden veröffentlicht die Funktion eine von zwei möglichen Nachrichten im Thema `hello/world`.
 - Der Ordner `greengrasssdk`. Dies ist das SDK.
3. Kopieren Sie den Ordner `greengrasssdk` in den Ordner `HelloWorld`, der `greengrassHelloWorld.py` enthält.
 4. Um das Lambda-Funktions-Deployment-Paket zu erstellen, speichern Sie `greengrassHelloWorld.py` und den `greengrasssdk` Ordner in einer komprimierten `zip` Datei mit dem Namen `hello_world_python_lambda.zip`. Die Datei `py` und der Ordner `greengrasssdk` müssen sich im Root des Verzeichnisses befinden.



Auf UNIX-ähnlichen Systemen (einschließlich des Mac-Terminals) können Sie den folgenden Befehl verwenden, um die Datei und den Ordner zu verpacken:

```
zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
```

Note

Je nach Ihrer Verteilung müssen Sie möglicherweise zuerst `zip` installieren (z. B. durch Ausführen von `sudo apt-get install zip`). Der Installationsbefehl kann für Ihre Verteilung abweichen.

Jetzt sind Sie bereit, Ihre Lambda-Funktion zu erstellen und das Bereitstellungspaket hochzuladen.

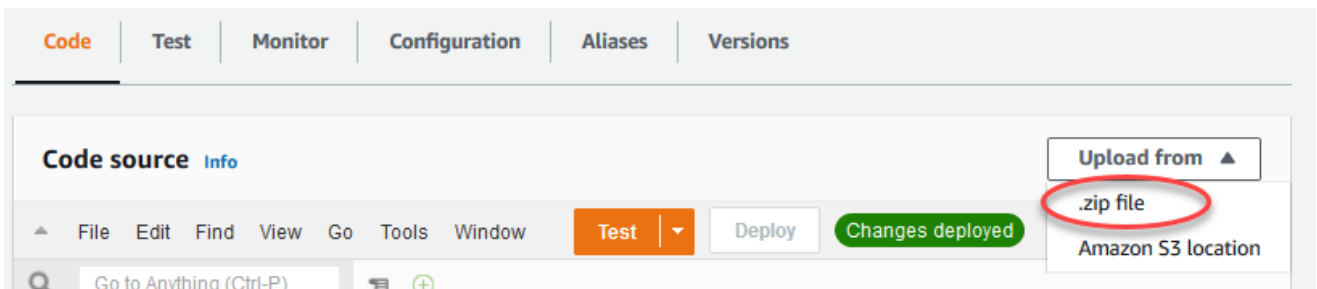
5. Öffnen Sie die Lambda-Konsole und wählen Sie `Create function`.
6. Wählen Sie `Von Grund auf neu schreiben aus`.
7. Geben Sie Ihrer Funktion den Namen **Greengrass_HelloWorld** und richten Sie die verbleibenden Felder wie folgt ein:
 - Wählen Sie für `Runtime (Laufzeit)` die Option `Python 3.7` aus.

- Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.

Wählen Sie Funktion erstellen aus.

8. Laden Sie das Bereitstellungspaket Ihrer Lambda-Funktion hoch:

- a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Hochladen und dann Ihr `hello_world_python_lambda.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Geben Sie unter Handler **`greengrassHelloWorld.function_handler`** ein.



- d. Wählen Sie Save (Speichern) aus.

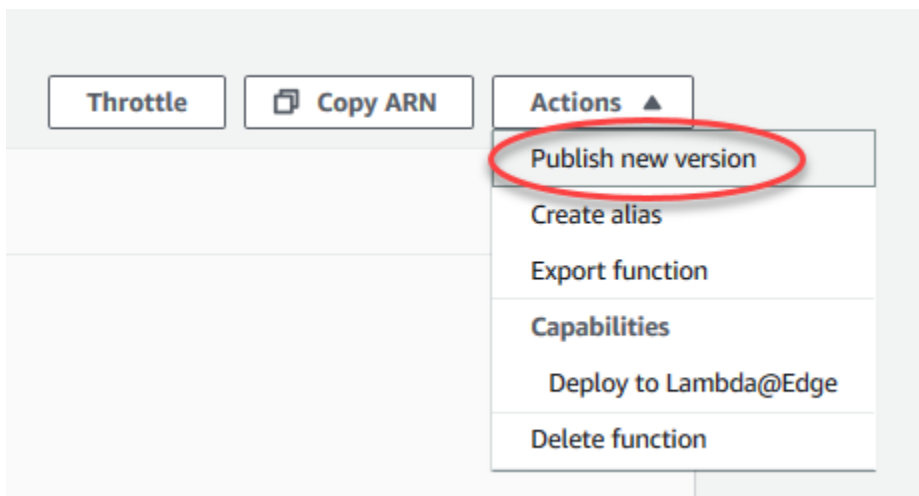
Note

Die Test-Schaltfläche auf der AWS Lambda Konsole funktioniert mit dieser Funktion nicht. Das AWS IoT Greengrass Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig in der AWS Lambda Konsole auszuführen. Diese Module (zum Beispiel `greengrass_common`) werden den Funktionen zur Verfügung gestellt, nachdem sie auf Ihrem Greengrass-Kern bereitgestellt wurden.

9.

Veröffentlichen Sie die Lambda-Funktion:

- Wählen Sie im Menü Aktionen oben auf der Seite die Option Neue Version veröffentlichen aus.



- Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.

Publish new version from \$LATEST ×

Publishing a new version will save a "snapshot" of the code and configuration of the \$LATEST version. You will be unable to edit the new version's code. Please click to confirm.

Version description

Cancel **Publish**

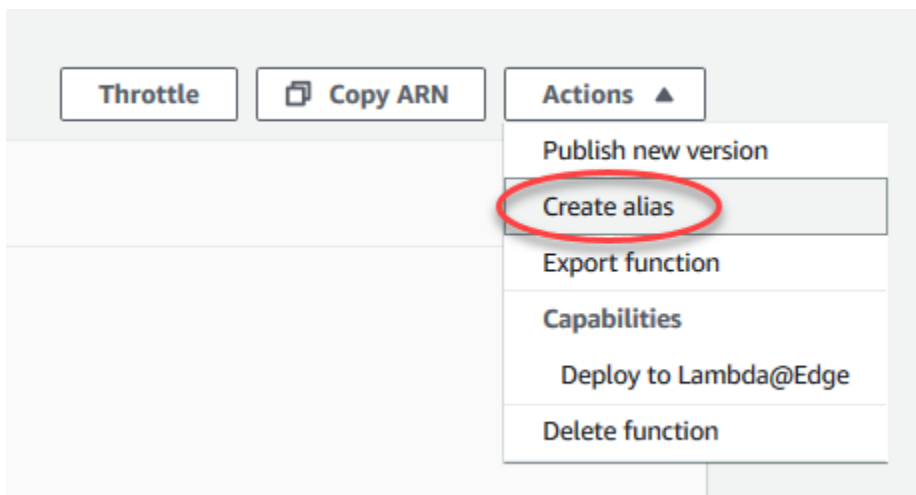
10.

Erstellen Sie einen [Alias](#) für die [Lambda-Funktionsversion](#):

Note

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

- a. Wählen Sie im Menü Aktionen oben auf der Seite die Option Alias erstellen aus.



- b. Geben Sie dem Alias einen Namen **GG_HelloWorld**, legen Sie die Version fest **1** (was der Version entspricht, die Sie gerade veröffentlicht haben), und wählen Sie dann Speichern aus.

Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

Description - optional

Version

1 ▼

► Weighted alias

Cancel Save

Konfigurieren Sie die Lambda-Funktion für AWS IoT Greengrass

Sie sind jetzt bereit, Ihre Lambda-Funktion für AWS IoT Greengrass zu konfigurieren.

In diesem Schritt führen Sie folgende Aktionen aus:

- Verwenden Sie die AWS IoT Konsole, um die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzuzufügen.
- Konfigurieren Sie gruppenspezifische Einstellungen für die Lambda-Funktion.
- Fügen Sie der Gruppe ein Abonnement hinzu, in dem die Lambda-Funktion MQTT-Nachrichten veröffentlichen kann. AWS IoT
- Konfigurieren Sie lokale Protokolleinstellungen für die Gruppe.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie unter Greengrass-Gruppen die Gruppe aus, die Sie in [Modul 2](#) erstellt haben.

3. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus, scrollen Sie dann nach unten zum Abschnitt Meine Lambda-Funktionen und wählen Sie Lambda-Funktion hinzufügen aus.
4. Wählen Sie den Namen der Lambda-Funktion aus, die Sie im vorherigen Schritt erstellt haben (Greengrass_ HelloWorld, nicht den Aliasnamen).
5. Wählen Sie für die Version Alias: GG_ HelloWorld
6. Nehmen Sie im Abschnitt Lambda-Funktionskonfiguration die folgenden Änderungen vor:
 - Stellen Sie für den Systembenutzer und die Gruppe die Option Gruppenstandard verwenden ein.
 - Setzen Sie die Containerisierung der Lambda-Funktion auf Gruppenstandard verwenden.
 - Legen Sie für Timeout 25 Sekunden fest. Diese Lambda-Funktion ruht vor jedem Aufruf für 5 Sekunden.
 - Wählen Sie für Pinned die Option True aus.

 Note

Eine langlebige (oder fixierte) Lambda-Funktion wird nach dem Start automatisch AWS IoT Greengrass gestartet und läuft in einem eigenen Container weiter. Dies steht im Gegensatz zu einer On-Demand-Lambda-Funktion, die startet, wenn sie aufgerufen wird, und stoppt, wenn keine Aufgaben mehr zur Ausführung übrig sind. Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

7. Wählen Sie Lambda-Funktion hinzufügen, um Ihre Änderungen zu speichern. Hinweise zu den Eigenschaften von Lambda-Funktionen finden Sie unter [the section called “Steuerung der Ausführung von Greengrass Lambda-Funktionen”](#).

Erstellen Sie als Nächstes ein Abonnement, an das die Lambda-Funktion [MQTT-Nachrichten senden kann](#). AWS IoT Core

Eine Greengrass Lambda-Funktion kann MQTT-Nachrichten austauschen mit:

- [Geräte](#) in der Greengrass-Gruppe.
- [Konnektoren](#) in der Gruppe.

- Andere Lambda-Funktionen in der Gruppe.
- AWS IoT Core.
- Service des lokalen Schattens. Weitere Informationen finden Sie unter [the section called “Modul 5: Interaktion mit Geräteschatten”](#).

Die Gruppe verwendet Abonnements, um zu steuern, wie diese Entitäten miteinander kommunizieren können sollen. Abonnements bieten vorhersehbare Interaktionen und eine Sicherheitsebene.

Ein Abonnement besteht aus einer Quelle, einem Ziel und einem Thema. Die Quelle beschreibt den Ursprung der Nachricht. Das Ziel beschreibt die Destination der Nachricht. Mithilfe des Themas können Sie die Daten filtern, die von der Quelle ans Ziel gesendet werden. Die Quelle oder das Ziel kann ein Greengrass-Gerät, eine Lambda-Funktion, ein Connector, ein Geräteshadow oder sein. AWS IoT Core

 Note

Ein Abonnement wird in dem Sinne weitergeleitet, dass Nachrichten in eine bestimmte Richtung fließen: von der Quelle zum Ziel. Wenn Sie eine Zwei-Wege-Kommunikation ermöglichen möchten, müssen Sie zwei Abonnements einrichten.

 Note

Derzeit erlaubt der Themenfilter für Abonnements nicht mehr als ein einzelnes + Zeichen in einem Thema. Der Themenfilter erlaubt nur ein einzelnes # Zeichen am Ende eines Themas.

Die Greengrass_HelloWorld Lambda-Funktion sendet Nachrichten nur an das hello/world Thema in AWS IoT Core, sodass Sie nur ein Abonnement von der Lambda-Funktion bis erstellen müssen. AWS IoT Core Sie erstellen dies im nächsten Schritt.

8. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.

Ein Beispiel, das Ihnen zeigt, wie Sie ein Abonnement mit dem erstellen AWS CLI, finden Sie [create-subscription-definition](#) in der AWS CLI Befehlsreferenz.

9. Wählen Sie im Quelltyp die Lambda-Funktion und für die Quelle die Option HelloWorldGreengrass_ aus.
10. Wählen Sie für den Zieltyp Service und für das Ziel IoT Cloud aus.
11. Geben Sie **hello/world** als Themenfilter Folgendes ein und wählen Sie dann Abonnement erstellen aus.
12. Konfigurieren Sie die Einstellungen für die Protokollierung der Gruppe. In diesem Tutorial konfigurieren Sie AWS IoT Greengrass Systemkomponenten und benutzerdefinierte Lambda-Funktionen, um Protokolle in das Dateisystem des Kerngeräts zu schreiben.
 - a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Logs aus.
 - b. Wählen Sie im Abschnitt Konfiguration der lokalen Protokolle die Option Bearbeiten aus.
 - c. Behalten Sie im Dialogfeld Konfiguration der lokalen Protokolle bearbeiten die Standardwerte für Protokollebenen und Speichergrößen bei und wählen Sie dann Speichern aus.

Sie können Protokolle verwenden, um Probleme zu beheben, die während dieses Tutorials möglicherweise auftreten. Wenn Sie Probleme beheben, können Sie die Protokollierungsstufe vorübergehend zu Debug ändern. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Dateisystemprotokolle”](#).

13. Wenn die Java 8-Laufzeitumgebung nicht auf Ihrem Core-Gerät installiert ist, müssen Sie sie installieren oder den Stream-Manager deaktivieren.

 Note

In diesem Tutorial wird Stream-Manager nicht verwendet, jedoch wird der Workflow zur Erstellung von Standardgruppen verwendet, der Stream-Manager standardmäßig aktiviert. Wenn Stream-Manager aktiviert ist, Java 8 jedoch nicht installiert ist, schlägt die Gruppenbereitstellung fehl. Weitere Informationen finden Sie in den [Anforderungen für Stream-Manager](#).

So deaktivieren Sie Stream-Manager:

- a. Wählen Sie auf der Seite mit den Gruppeneinstellungen die Registerkarte Lambda-Funktionen aus.
- b. Wählen Sie im Abschnitt System Lambda functions die Option Stream Manager und dann Bearbeiten aus.
- c. Wählen Sie Disable (Deaktivieren) und anschließend Save (Speichern) aus.

Bereitstellen von Cloud-Konfigurationen auf einem Greengrass-Core-Gerät

1. Stellen Sie sicher, dass Ihr Greengrass-Core-Gerät mit dem Internet verbunden ist. Versuchen Sie dazu beispielsweise, zu einer Webseite zu navigieren.
2. Stellen Sie sicher, dass der Greengrass-Daemon auf Ihrem Core-Gerät ausgeführt wird. Führen Sie in Ihrem Core-Device-Terminal die folgenden Befehle aus, um zu überprüfen, ob der Daemon läuft, und starten Sie ihn bei Bedarf.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/1.11.6/bin/daemon` enthält, dann wird der Daemon ausgeführt.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

Jetzt sind Sie bereit, die Lambda-Funktion und Abonnementkonfigurationen auf Ihrem Greengrass-Core-Gerät bereitzustellen.

3. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
4. Wählen Sie unter Greengrass-Gruppen die Gruppe aus, die Sie in [Modul 2](#) erstellt haben.
5. Wählen Sie auf der Gruppenkonfigurationsseite Deploy aus.
6. Wählen Sie auf der Registerkarte Lambda-Funktionen im Abschnitt System-Lambda-Funktionen die Option IP-Detektor aus.

7. Wählen Sie Bearbeiten und wählen Sie MQTT-Broker-Endpunkte automatisch erkennen und überschreiben aus. Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Die automatische Ermittlung wird empfohlen, aber AWS IoT Greengrass unterstützt auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

Der erste Bereitstellungsvorgang kann einige Minuten dauern. Nachdem die Bereitstellung abgeschlossen ist, sollten Sie Successfully completed (Erfolgreich abgeschlossen) in der Spalte Status der Seite Deployments (Bereitstellungen) sehen:

 Note

Der Bereitstellungsstatus wird ebenfalls im Header der Seite unter dem Namen der Gruppe angezeigt.

Hilfe zur Problembeseitigung finden Sie unter [Fehlerbehebung](#).

Stellen Sie sicher, dass die Lambda-Funktion auf dem Kerngerät ausgeführt wird

1. Wählen Sie im Navigationsbereich der [AWS IoT Konsole](#) unter Test die Option MQTT-Testclient aus.
2. Wählen Sie die Registerkarte Thema abonnieren.
3. Rufen Sie **hello/world** den Themenfilter auf und erweitern Sie die Zusätzliche Konfiguration.
4. Geben Sie die in jedes der folgenden Felder aufgelisteten Informationen ein:
 - Wählen Sie für Servicequalität 0 aus.
 - Wählen Sie für MQTT-Nutzlast-Anzeige die Option Nutzlasten als Zeichenfolgen anzeigen aus.
5. Wählen Sie Subscribe (Abonnieren) aus.

Unter der Annahme, dass die Lambda-Funktion auf Ihrem Gerät ausgeführt wird, veröffentlicht sie Meldungen, die den folgenden ähneln, zum hello/world Thema:

Subscriptions

hello/world  

hello/world

April 29, 2021, 17:35:40 (UTC-0400)

```
{
  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-
v7+-armv7l-with-debian-8.0"
}
```

Pause

Clear

Export

Edit

Obwohl die Lambda-Funktion weiterhin MQTT-Nachrichten an das hello/world Thema sendet, stoppen Sie den AWS IoT Greengrass Daemon nicht. Bei den verbleibenden Modulen wird nämlich davon ausgegangen, dass die Funktion ausgeführt wird.

Sie können die Funktion und das Abonnement aus der Gruppe löschen:

- Wählen Sie auf der Gruppenkonfigurationsseite unter der Registerkarte Lambda-Funktionen die Lambda-Funktion aus, die Sie entfernen möchten, und wählen Sie Entfernen aus.
- Wählen Sie auf der Gruppenkonfigurationsseite unter der Registerkarte Abonnements das Abonnement aus und klicken Sie dann auf Löschen.

Die Funktion und das Abonnement werden während der nächsten Gruppenbereitstellung vom Kern entfernt.

Modul 3 (Teil 2): Lambda-Funktionen auf AWS IoT Greengrass

In diesem Modul werden die Unterschiede zwischen On-Demand-Funktionen und langlebigen Lambda-Funktionen untersucht, die AWS IoT Greengrass auf dem Core ausgeführt werden.

Führen Sie vor Beginn das [Greengrass Device Setup-Skript](#) aus oder stellen Sie sicher, dass Sie [Modul 1](#), [Modul 2](#) und [Modul 3 \(Teil 1\)](#) abgeschlossen haben.

Dieses Modul sollte etwa 30 Minuten in Anspruch nehmen.

Themen

- [Lambda-Funktion erstellen und verpacken](#)
- [Konfigurieren Sie langlebige Lambda-Funktionen für AWS IoT Greengrass](#)

- [Testen Sie langlebige Lambda-Funktionen](#)
- [Testen Sie Lambda-Funktionen auf Abruf](#)

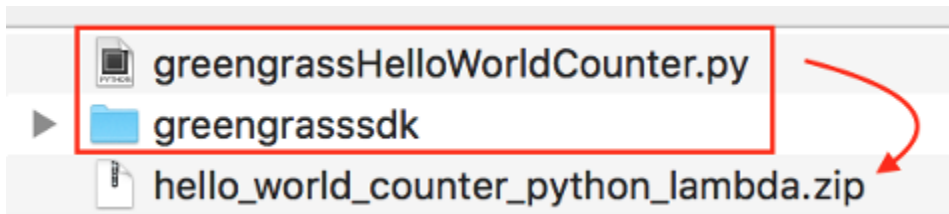
Lambda-Funktion erstellen und verpacken

In diesem Schritt führen Sie folgende Aktionen aus:

- Erstellen Sie ein Paket zur Bereitstellung von Lambda-Funktionen, das den Funktionscode und die Abhängigkeiten enthält.
 - Verwenden Sie die Lambda-Konsole, um eine Lambda-Funktion zu erstellen und das Bereitstellungspaket hochzuladen.
 - Veröffentlichen Sie eine Version der Lambda-Funktion und erstellen Sie einen Alias, der auf die Version verweist.
1. Rufen Sie auf Ihrem Computer das AWS IoT Greengrass Core SDK für Python auf, das Sie [the section called “Eine Lambda-Funktion erstellen und verpacken”](#) in Modul 3-1 heruntergeladen und extrahiert haben.

Die Lambda-Funktion in diesem Modul verwendet:

- Die Datei `greengrassHelloWorldCounter.py` in `examples\HelloWorldCounter`. Dies ist Ihr Lambda-Funktionscode.
 - Der Ordner `greengrasssdk`. Dies ist das SDK.
2. Erstellen Sie ein Paket zur Bereitstellung von Lambda-Funktionen:
 - a. Kopieren Sie den Ordner `greengrasssdk` in den Ordner `HelloWorldCounter`, der `greengrassHelloWorldCounter.py` enthält.
 - b. Speichern Sie `greengrassHelloWorldCounter.py` und den Ordner `greengrasssdk` in einer zip-Datei mit der Bezeichnung `hello_world_counter_python_lambda.zip`. Die Datei `py` und der Ordner `greengrasssdk` müssen sich im Root des Verzeichnisses befinden.



Auf UNIX-ähnlichen Systemen (einschließlich des Mac-Terminals), auf denen zip installiert ist, können Sie den folgenden Befehl verwenden, um die Datei und den Ordner zu verpacken:

```
zip -r hello_world_counter_python_lambda.zip greengrasssdk  
greengrassHelloWorldCounter.py
```

Jetzt sind Sie bereit, Ihre Lambda-Funktion zu erstellen und das Bereitstellungspaket hochzuladen.

3. Öffnen Sie die Lambda-Konsole und wählen Sie Create function.
4. Wählen Sie Von Grund auf neu schreiben aus.
5. Geben Sie Ihrer Funktion den Namen **Greengrass_HelloWorld_Counter** und richten Sie die verbleibenden Felder wie folgt ein:
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass. Oder Sie können die Rolle, die Sie in Modul 3-1 erstellt haben, wiederverwenden.

Wählen Sie Funktion erstellen aus.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Python 3.7 ▼

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

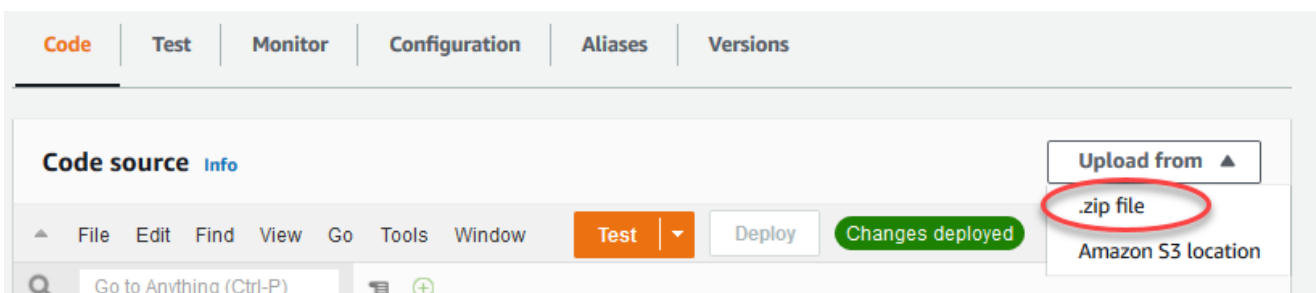
► **Change default execution role**

► **Advanced settings**

Cancel **Create function**

6. Laden Sie Ihr Lambda-Funktionsbereitstellungspaket hoch.

- Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



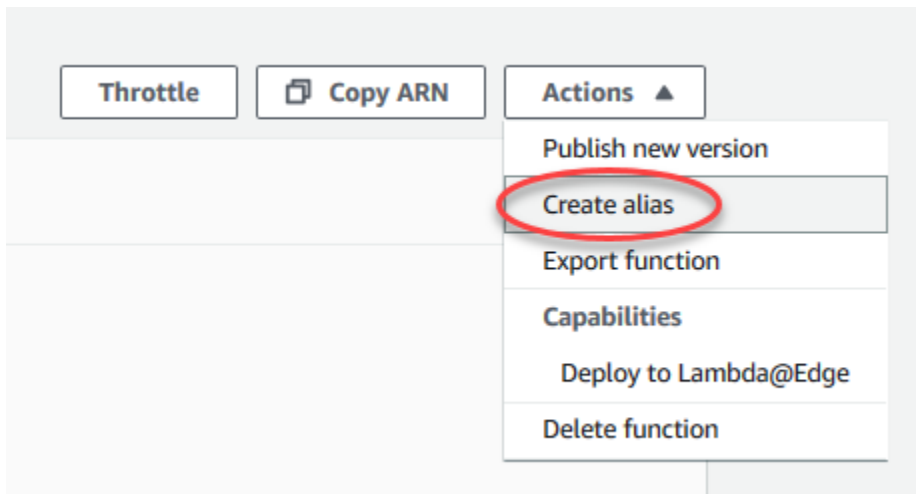
- Wählen Sie Hochladen und dann Ihr `hello_world_counter_python_lambda.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

- Geben Sie unter Handler **greengrassHelloWorldCounter.function_handler** ein.
- d. Wählen Sie Save (Speichern) aus.

Note

Die Testtaste auf der AWS Lambda Konsole funktioniert mit dieser Funktion nicht. Das AWS IoT Greengrass Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig in der AWS Lambda Konsole auszuführen. Diese Module (zum Beispiel `greengrass_common`) werden den Funktionen zur Verfügung gestellt, nachdem sie auf Ihrem Greengrass-Kern bereitgestellt wurden.

7. Veröffentlichen Sie die erste Version der Funktion.
- a. Wählen Sie im Menü Aktionen oben auf der Seite die Option Neue Version veröffentlichen aus. Geben Sie für Version description (Versionsbeschreibung) **First version** ein.
 - b. Wählen Sie Publish.
8. Erstellen Sie einen Alias für die Funktionsversion.
- a. Wählen Sie oben auf der Seite im Menü Aktionen die Option Alias erstellen aus.



- b. Geben Sie unter Name **GG_HW_Counter** ein.
- c. Wählen Sie für Version die Option 1.
- d. Wählen Sie Save (Speichern) aus.

Create alias

Alias configuration

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name

Description - optional

Version

1 ▼

► Weighted alias

CancelSave

Aliase bilden eine einzige Entität für Ihre Lambda-Funktion, die Greengrass-Geräte abonnieren können. Auf diese Weise müssen Sie Abonnements nicht jedes Mal mit neuen Versionsnummern der Lambda-Funktion aktualisieren, wenn die Funktion geändert wird.

Konfigurieren Sie langlebige Lambda-Funktionen für AWS IoT Greengrass

Sie sind jetzt bereit, Ihre Lambda-Funktion für AWS IoT Greengrass zu konfigurieren.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie unter Greengrass-Gruppen die Gruppe aus, die Sie in [Modul 2](#) erstellt haben.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen und dann unter Meine Lambda-Funktionen die Option Hinzufügen aus.
4. Wählen Sie für die Lambda-Funktion HelloWorld Greengrass_ _Counter.
5. Wählen Sie für die Lambda-Funktionsversion den Alias für die Version aus, die Sie veröffentlicht haben.

6. Geben Sie für Timeout (Sekunden) den Wert ein. **25** Diese Lambda-Funktion ruht vor jedem Aufruf 20 Sekunden lang.
7. Wählen Sie für Pinned die Option True aus.
8. Behalten Sie die Standardwerte für alle anderen Felder bei und wählen Sie Lambda-Funktion hinzufügen.

Testen Sie langlebige Lambda-Funktionen

Eine [langlebige](#) Lambda-Funktion wird automatisch gestartet, wenn der AWS IoT Greengrass Kern in einem einzigen Container (oder einer Sandbox) gestartet und ausgeführt wird. Alle Variablen und sämtliche Vorverarbeitungslogik, die außerhalb des Funktionshandlers definiert sind, werden für jeden Aufruf des Funktionshandlers beibehalten. Mehrere Aufrufe werden so lange in eine Warteschlange gesetzt, bis die Ausführung vorheriger Aufrufe abgeschlossen ist.

Der in diesem Modul verwendete `greengrassHelloWorldCounter.py`-Code definiert eine `my_counter`-Variable außerhalb des Funktionshandlers.

Note

Sie können den Code in der AWS Lambda Konsole oder im [AWS IoT Greengrass Core SDK für Python](#) unter anzeigen GitHub.

In diesem Schritt erstellen Sie Abonnements, die die Lambda-Funktion und AWS IoT den Austausch von MQTT-Nachrichten ermöglichen. Im Anschluss daran stellen Sie die Gruppe bereit und testen die Funktion.

1. Wählen Sie auf der Gruppenkonfigurationsseite Abonnements und dann Hinzufügen aus.
2. Wählen Sie unter Quelltyp die Option Lambda-Funktion und dann HelloWorldGreengrass_Counter aus.
3. Wählen Sie unter Zieltyp die Option Service und dann IoT Cloud aus.
4. Geben Sie für Topic filter (Themenfilter) die Zeichenfolge **hello/world/counter** ein.
5. Wählen Sie Create subscription (Abonnement erstellen) aus.

Dieses einzelne Abonnement geht nur in eine Richtung: von der Greengrass_HelloWorld_Counter Lambda-Funktion bis AWS IoT. Um diese Lambda-

Funktion aus der Cloud aufzurufen (oder auszulösen), müssen Sie ein Abonnement in umgekehrter Richtung erstellen.

6. Folgen Sie den Schritten 1 bis 5, um ein weiteres Abonnement hinzuzufügen, das die folgenden Werte verwendet. Dieses Abonnement ermöglicht es der Lambda-Funktion, Nachrichten von AWS IoT zu empfangen. Sie verwenden dieses Abonnement, wenn Sie eine Nachricht von der AWS IoT Konsole senden, die die Funktion aufruft.
 - Wählen Sie als Quelle Service und dann IoT Cloud aus.
 - Wählen Sie für das Ziel Lambda-Funktion und dann HelloWorldGreengrass_ _Counter aus.
 - Geben Sie für den Themenfilter **hello/world/counter/trigger** ein.

In diesem Themenfilter wird die Erweiterung `/trigger` verwendet, da Sie zwei Abonnements erstellt haben, die sich nicht gegenseitig beeinträchtigen sollen.

7. Stellen Sie sicher, dass der Greengrass-Daemon läuft, wie unter beschrieben. [Bereitstellen von Cloud-Konfigurationen für ein Core-Gerät](#)
8. Wählen Sie auf der Gruppenkonfigurationsseite Deploy aus.
9. Kehren Sie nach Abschluss der Bereitstellung zur Startseite der AWS IoT Konsole zurück und wählen Sie Test aus.
10. Konfigurieren Sie die folgenden Felder:
 - Geben Sie für Abonnementthema **hello/world/counter** ein.
 - Wählen Sie für Servicequalität 0 aus.
 - Wählen Sie für MQTT-Nutzlast-Anzeige die Option Nutzlasten als Zeichenfolgen anzeigen aus.
11. Wählen Sie Subscribe (Abonnieren) aus.

Anders als in [Teil 1](#) dieses Moduls sollten Sie nach dem Abonnement von `hello/world/counter` keine Nachrichten sehen. Dies liegt daran, dass der `greengrassHelloWorldCounter.py` Code, der im `hello/world/counter`-Thema veröffentlicht, sich innerhalb des Funktionshandlers befindet, der nur ausgeführt wird, wenn die Funktion aufgerufen wird.

In diesem Modul haben Sie die `Greengrass_HelloWorld_Counter` Lambda-Funktion so konfiguriert, dass sie aufgerufen wird, wenn sie eine MQTT-Nachricht zu diesem Thema empfängt. `hello/world/counter/trigger`

Das Greengrass_HelloWorld_Counter to IoT Cloud-Abonnement ermöglicht es der Funktion, Nachrichten zum Thema AWS IoT an zu senden. hello/world/counter Das Abonnement IoT Cloud to Greengrass_HelloWorld_Counter ermöglicht das Senden von Nachrichten AWS IoT an die Funktion zu diesem Thema. hello/world/counter/trigger

- Um den langlebigen Lebenszyklus zu testen, rufen Sie die Lambda-Funktion auf, indem Sie eine Nachricht zum Thema veröffentlichen. hello/world/counter/trigger Sie können die Standardnachricht verwenden.

Subscribe to a topic | **Publish to a topic**

Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Message payload

Additional configuration

Publish

Note

Die Greengrass_HelloWorld_Counter-Funktion ignoriert den Inhalt der empfangenen Nachrichten. Sie führt lediglich den Code im `function_handler` aus, wodurch eine Nachricht an das hello/world/counter-Thema gesendet wird. Sie können diesen Code im [AWS IoT Greengrass Core SDK für Python](#) unter überprüfen GitHub.

Jedes Mal, wenn eine Nachricht im Thema hello/world/counter/trigger veröffentlicht wird, wird die `my_counter`-Variable um einen Schritt erhöht. Diese Anzahl von Aufrufen wird in den Nachrichten angezeigt, die von der Lambda-Funktion gesendet werden. Da der Funktionshandler einen 20-Sekunden-Schlafzyklus (`time.sleep(20)`) beinhaltet, werden beim wiederholten Auslösen des Handlers Antworten vom Kern in eine Warteschlange gestellt. AWS IoT Greengrass

The screenshot shows the 'Subscriptions' page in the AWS IoT Greengrass console. On the left, a sidebar lists the subscription 'hello/world/counter' with a heart icon and a close button. The main area displays three invocations of this subscription, each with a dropdown arrow, the name 'hello/world/counter', and a timestamp. The invocation details are shown in a JSON format. The 'Invocation Count' is highlighted in red circles in each entry.

Subscription	Timestamp	Message	Invocation Count
hello/world/counter	May 03, 2021, 10:05:00 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }</pre>	3
hello/world/counter	May 03, 2021, 10:04:40 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }</pre>	2
hello/world/counter	May 03, 2021, 10:04:20 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0" }</pre>	1

Testen Sie Lambda-Funktionen auf Abruf

Eine [On-Demand-Lambda-Funktion](#) ähnelt in ihrer Funktionalität einer Cloud-basierten AWS Lambda Funktion. Mehrere Aufrufe einer On-Demand-Lambda-Funktion können parallel ausgeführt werden. Ein Aufruf der Lambda-Funktion erstellt einen separaten Container zur Verarbeitung von Aufrufen oder verwendet einen vorhandenen Container wieder, sofern die Ressourcen dies zulassen. Variablen oder Vorverarbeitungen, die außerhalb des Funktionshandlers definiert sind, werden nicht beibehalten, wenn Container erstellt werden.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
2. Wählen Sie unter Meine Lambda-Funktionen die Greengrass_HelloWorld_CounterLambda-Funktion.
3. Auf der Greengrass_HelloWorld_CounterWählen Sie auf der Detailseite Bearbeiten aus.

4. Wählen Sie für Fixiert die Option Falsch und dann Speichern aus.
5. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Bereitstellen aus.
6. Kehren Sie nach Abschluss der Bereitstellung zur Startseite der AWS IoT Konsole zurück und wählen Sie Test aus.
7. Konfigurieren Sie die folgenden Felder:
 - Geben Sie für Abonnementthema **hello/world/counter** ein.
 - Wählen Sie für Servicequalität 0 aus.
 - Wählen Sie für MQTT-Nutzlast-Anzeige die Option Nutzlasten als Zeichenfolgen anzeigen aus.

Subscribe to a topic

Publish to a topic

Topic filter [Info](#)

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

hello/world/counter

▼ Additional configuration

Number of messages to keep

The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

100

Quality of service

When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once

☐ Quality of Service 1 - Message will be delivered at least once

MQTT payload display

☐ Auto-format JSON payloads (improves readability)

☒ Display payloads as strings (more accurate)

☐ Display raw payloads (displays binary data as hexadecimal values)

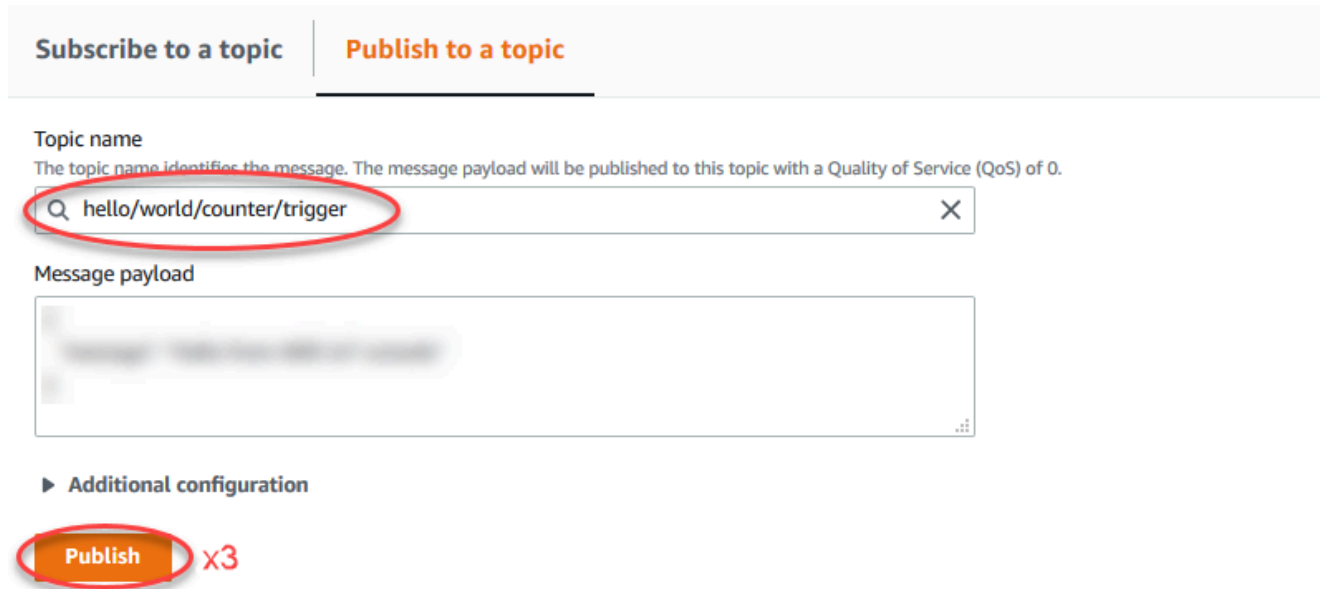
Subscribe

8. Wählen Sie Subscribe (Abonnieren) aus.

Note

Sie sollten nach dem Abonnement keine Nachrichten sehen.

9. Um den On-demand-Lebenszyklus zu überprüfen, rufen Sie die Funktion auf, indem Sie eine Nachricht im `hello/world/counter/trigger`-Thema veröffentlichen. Sie können die Standardnachricht verwenden.
 - a. Wählen Sie dreimal schnell, innerhalb von fünf Sekunden nach jedem Drücken der Taste, Veröffentlichen.



The screenshot shows the AWS IoT Greengrass console interface for publishing a message. At the top, there are two tabs: 'Subscribe to a topic' and 'Publish to a topic'. The 'Publish to a topic' tab is selected. Below the tabs, there is a 'Topic name' section with a text input field containing 'hello/world/counter/trigger'. A red circle highlights this input field. Below the topic name is a 'Message payload' section with a large text area. Underneath, there is a section titled 'Additional configuration'. At the bottom of this section, there is a red 'Publish' button, which is also circled in red, followed by a red 'x3' indicating that the button should be clicked three times.

Jede Veröffentlichung ruft den Funktionshandler auf und erstellt einen neuen Container für jeden Aufruf. Die Anzahl der Aufrufe wird nicht erhöht, wenn Sie die Funktion dreimal ausgelöst haben, da jede On-Demand-Lambda-Funktion über einen eigenen Container/eine eigene Sandbox verfügt.

The screenshot shows the AWS IoT Greengrass interface. On the left, under the 'Subscriptions' tab, the subscription 'hello/world/counter' is selected. On the right, the 'hello/world/counter' topic is displayed with buttons for 'Pause', 'Clear', 'Export', and 'Edit'. Below the topic name, three messages are listed, each with a timestamp and a JSON payload. In each message, the 'Invocation Count: 1' is circled in red.

Subscription	Topic	Timestamp	Message
hello/world/counter	hello/world/counter	May 03, 2021, 10:14:13 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>
hello/world/counter	hello/world/counter	May 03, 2021, 10:14:12 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>
hello/world/counter	hello/world/counter	May 03, 2021, 10:14:11 (UTC-0400)	<pre>{ "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0. Invocation Count: 1" }</pre>

- b. Wählen Sie nach ungefähr 30 Sekunden die Option Publish to topic (Veröffentlichung im Thema) aus. Der Zahl der Aufrufe sollte sich auf 2 erhöhen. Dies zeigt, dass ein Container, der durch einen vorherigen Aufruf erstellt wurde, erneut verwendet wird, und Variablen, die außerhalb des Funktionshandlers vorab verarbeitet wurden, gespeichert wurden.

The screenshot displays the AWS IoT Greengrass console interface. On the left, a sidebar titled 'Subscriptions' lists the subscription 'hello/world/counter' with a heart and close icon. The main panel shows the details for this subscription, titled 'hello/world/counter', with buttons for 'Pause', 'Clear', 'Export', and 'Edit'. Two event logs are visible:

- Event 1:** Timestamped 'May 03, 2021, 10:14:40 (UTC-0400)'. The message body is a JSON object:

```
{  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0."}
```

 The text 'Invocation Count: 2' is circled in red.
- Event 2:** Timestamped 'May 03, 2021, 10:14:11 (UTC-0400)'. The message body is a JSON object:

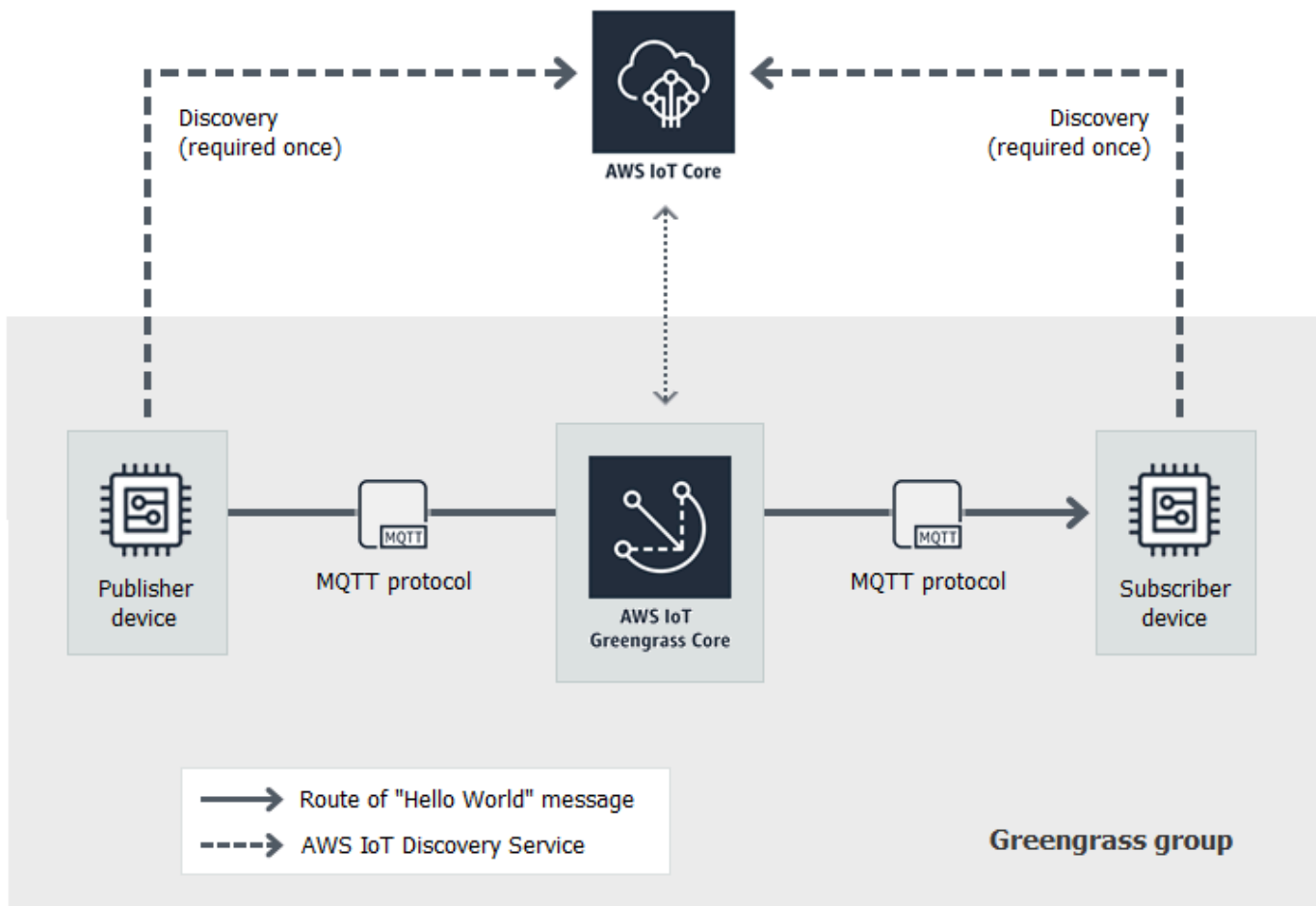
```
{  "message": "Hello world! Sent from Greengrass Core running on platform: Linux-4.9.30-v7+-armv7l-with-Debian-8.0."}
```

 The text 'Invocation Count: 1' is circled in red.

Sie sollten jetzt die beiden Arten von Lambda-Funktionen verstehen, die auf dem AWS IoT Greengrass Kern ausgeführt werden können. Das nächste Modul, [Modul 4](#), zeigt Ihnen, wie lokale IoT-Geräte in einer AWS IoT Greengrass Gruppe interagieren können.

Modul 4: Interaktion mit Client-Geräten in einer AWS IoT Greengrass Gruppe

Dieses Modul zeigt Ihnen, wie lokale IoT-Geräte, sogenannte Client-Geräte oder Geräte, eine Verbindung zu einem AWS IoT Greengrass Kerngerät herstellen und mit diesem kommunizieren können. Client-Geräte, die eine Verbindung zu einem AWS IoT Greengrass Core herstellen, sind Teil einer AWS IoT Greengrass Gruppe und können am AWS IoT Greengrass Programmierparadigma teilnehmen. In diesem Modul sendet ein Client-Gerät eine Hello World-Nachricht an ein anderes Client-Gerät in der Greengrass-Gruppe.



Führen Sie vor Beginn das Skript [Greengrass Device Setup](#) aus oder schließen Sie [Modul 1](#) und [Modul 2](#) ab. Dieses Modul erstellt zwei simulierte Client-Geräte. Sie benötigen keine anderen Komponenten oder Geräte.

Dieses Modul sollte nicht mehr als 30 Minuten in Anspruch nehmen.

Themen

- [Client-Geräte in einer AWS IoT Greengrass Gruppe erstellen](#)
- [Konfigurieren von Abonnements](#)
- [Installieren Sie das AWS IoT Device SDK für Python](#)
- [Testen der Kommunikation](#)

Client-Geräte in einer AWS IoT Greengrass Gruppe erstellen

In diesem Schritt fügen Sie Ihrer Greengrass-Gruppe zwei Client-Geräte hinzu. Dieser Prozess umfasst die Registrierung der Geräte als AWS IoT Dinge und die Konfiguration von Zertifikaten und Schlüsseln, mit denen sie eine Verbindung herstellen AWS IoT Greengrass können.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option Client-Geräte und dann Associate aus.
4. Wählen Sie im Modal Ein Client-Gerät mit dieser Gruppe verknüpfen die Option Neues Objekt erstellen AWS IoT aus.

Die Seite „Dinge erstellen“ wird auf einer neuen Registerkarte geöffnet.

5. Wählen Sie auf der Seite „Dinge erstellen“ die Option „Einzelnes Ding erstellen“ und dann „Weiter“ aus.
6. Registrieren Sie dieses Client-Gerät auf der Seite „Eigenschaften des Objekts angeben“ als **HelloWorld_Publisher** und wählen Sie dann Weiter aus.
7. Wählen Sie auf der Seite Gerätezertifikat konfigurieren die Option Weiter aus.
8. Führen Sie auf der Seite Richtlinien an Zertifikat anhängen einen der folgenden Schritte aus:
 - Wählen Sie eine bestehende Richtlinie aus, die Berechtigungen gewährt, die Client-Geräte benötigen, und wählen Sie dann Ding erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit dem Core AWS Cloud und dem Core verwendet.

- Erstellen Sie eine neue Richtlinie, die Client-Geräteberechtigungen gewährt, und fügen Sie sie an. Gehen Sie wie folgt vor:
 - a. Wählen Sie Richtlinie erstellen aus.

Die Seite Create policy (Richtlinie erstellen) wird in einer neuen Registerkarte geöffnet.

- b. Führen Sie auf der Seite Create policy (Richtlinie erstellen) die folgenden Schritte aus:
 - i. Geben Sie als Richtlinienname einen Namen ein, der die Richtlinie beschreibt, z. **GreengrassV1ClientDevicePolicy B.**

- ii. Wählen Sie auf der Registerkarte Richtlinienerklärungen unter Richtliniendokument die Option JSON aus.
- iii. Geben Sie das folgende Richtliniendokument ein. Diese Richtlinie ermöglicht es dem Client-Gerät, Greengrass-Kerne zu erkennen und zu allen MQTT-Themen zu kommunizieren. Informationen darüber, wie Sie den Zugriff dieser Richtlinie einschränken können, finden Sie unter. [Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass](#)

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- iv. Wählen Sie Create (Erstellen) aus, um die Richtlinie zu erstellen.
- c. Kehren Sie zur Browser-Registerkarte zurück, auf der die Seite Richtlinien an Zertifikat anhängen geöffnet ist. Gehen Sie wie folgt vor:

- i. Wählen Sie in der Liste Richtlinien die Richtlinie aus, die Sie erstellt haben, z. GreengrassV1ClientDevicePolicyB.

Wenn Sie die Richtlinie nicht sehen, klicken Sie auf die Schaltfläche „Aktualisieren“.

- ii. Wählen Sie Objekt erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit dem Core AWS Cloud und dem Core verwendet.

9. Laden Sie im Modal Zertifikate und Schlüssel herunterladen die Zertifikate des Geräts herunter.

 Important

Laden Sie die Sicherheitsressourcen herunter, bevor Sie „Fertig“ wählen.

Gehen Sie wie folgt vor:

- a. Wählen Sie unter Gerätezertifikat die Option Herunterladen aus, um das Gerätezertifikat herunterzuladen.
- b. Wählen Sie unter Datei mit öffentlichem Schlüssel die Option Herunterladen aus, um den öffentlichen Schlüssel für das Zertifikat herunterzuladen.
- c. Wählen Sie unter Datei mit privatem Schlüssel die Option Herunterladen aus, um die Datei mit dem privaten Schlüssel für das Zertifikat herunterzuladen.
- d. Lesen Sie die [Serverauthentifizierung](#) im AWS IoT Entwicklerhandbuch und wählen Sie das entsprechende Root-CA-Zertifikat aus. Wir empfehlen die Verwendung von Amazon Trust Services (ATS) -Endpunkten und ATS-Root-CA-Zertifikaten. Wählen Sie unter Root-CA-Zertifikate die Option Herunterladen für ein Root-CA-Zertifikat aus.
- e. Wählen Sie Fertig aus.

Notieren Sie sich die Zertifikat-ID, die in den Dateinamen für das Gerätezertifikat und die Schlüssel häufig vorkommt. Sie benötigen sie später.

10. Kehren Sie zur Browser-Registerkarte zurück und öffnen Sie das Modal Ein Client-Gerät mit dieser Gruppe verknüpfen. Gehen Sie wie folgt vor:

- a. Wählen Sie als Namen AWS IoT das HelloWorld_PublisherDing aus, das Sie erstellt haben.

Wenn Sie das Ding nicht sehen, wählen Sie die Schaltfläche „Aktualisieren“.

b. Wählen Sie Associate aus.

11. Wiederholen Sie die Schritte 3 bis 10, um der Gruppe ein zweites Client-Gerät hinzuzufügen.

Benennen Sie dieses Client-Gerät **HelloWorld_Subscriber**. Laden Sie die Zertifikate und Schlüssel für dieses Client-Gerät auf Ihren Computer herunter. Notieren Sie sich auch hier die ID des Zertifikats, die häufig in den Dateinamen für das HelloWorld_Subscriber-Gerät vorkommt.

Sie sollten jetzt zwei Client-Geräte in Ihrer Greengrass-Gruppe haben:

- HelloWorld_Herausgeber
- HelloWorld_Abonent

12. Erstellen Sie auf Ihrem Computer einen Ordner für die Sicherheitsanmeldeinformationen dieser Client-Geräte. Kopieren Sie die Zertifikate und Schlüssel in diesen Ordner.

Konfigurieren von Abonnements

In diesem Schritt aktivieren Sie das HelloWorld_Publisher-Clientgerät, um MQTT-Nachrichten an das HelloWorld_Subscriber-Clientgerät zu senden.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Hinzufügen aus.
2. Gehen Sie auf der Seite Abonnement erstellen wie folgt vor, um das Abonnement zu konfigurieren:
 - a. Wählen Sie als Quelltyp die Option Client-Gerät und dann HelloWorld_Publisher aus.
 - b. Wählen Sie unter Zieltyp die Option Client-Gerät und dann HelloWorld_Abonent aus.
 - c. Geben Sie für Topic filter (Themenfilter) die Zeichenfolge **hello/world/pubsub** ein.

Note

Sie können Abonnements aus den früheren Modulen löschen. Wählen Sie auf der Abonnementseite der Gruppe die Abonnements aus, die Sie löschen möchten, und klicken Sie dann auf Löschen.

d. Wählen Sie Create subscription (Abonnement erstellen) aus.

3. Stellen Sie sicher, dass die automatische Erkennung aktiviert ist, damit der Greengrass Core eine Liste seiner IP-Adressen veröffentlichen kann. Client-Geräte verwenden diese Informationen, um den Core zu ermitteln. Gehen Sie wie folgt vor:
 - a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
 - b. Wählen Sie unter System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
 - c. Wählen Sie unter IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben und dann Speichern aus.
4. Stellen Sie sicher, dass der Greengrass-Daemon läuft, wie unter beschrieben. [Bereitstellen von Cloud-Konfigurationen für ein Core-Gerät](#)
5. Wählen Sie auf der Gruppenkonfigurationsseite Deploy aus.

Der Bereitstellungsstatus wird im Header der Seite unter dem Namen der Gruppe angezeigt. Um Einzelheiten zur Bereitstellung anzuzeigen, wählen Sie die Registerkarte Bereitstellungen.

Installieren Sie das AWS IoT Device SDK für Python

Client-Geräte können AWS IoT Device SDK for Python verwenden, um mit AWS IoT und AWS IoT Greengrass Core-Geräten zu kommunizieren (unter Verwendung der Programmiersprache Python). Weitere Informationen, einschließlich der Anforderungen, finden Sie in der [Readme-Datei AWS IoT Device SDK](#) für Python unter GitHub.

In diesem Schritt installieren Sie das SDK und erhalten die `basicDiscovery.py` Beispielfunktion, die von den simulierten Client-Geräten auf Ihrem Computer verwendet wird.

1. Um das SDK mit allen erforderlichen Komponenten auf Ihrem Computer zu installieren, wählen Sie Ihr Betriebssystem aus:

Windows

1. Öffnen Sie eine [erweiterte Eingabeaufforderung](#) und führen Sie den folgenden Befehl aus:

```
python --version
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer für Python 2 kleiner als 2.7 oder für Python 3 kleiner als 3.3 ist,

installieren Sie Python 2.7+ oder Python 3.3+ gemäß den Anweisungen unter [Downloading Python](#). Weitere Informationen finden Sie unter [Using Python on Windows](#).

2. Laden Sie die [AWS IoT Device SDK für Python](#) als zip Datei herunter und extrahieren Sie sie an einen geeigneten Ort auf Ihrem Computer.

Notieren Sie sich den Dateipfad zum extrahierten `aws-iot-device-sdk-python-master`-Ordner, der die Datei `setup.py` enthält. Im nächsten Schritt wird dieser Dateipfad durch angezeigt *path-to-SDK-folder*.

3. Führen Sie von der erweiterten Eingabeaufforderung aus Folgendes aus:

```
cd path-to-SDK-folder
python setup.py install
```

macOS

1. Öffnen Sie ein Terminalfenster und führen Sie den folgenden Befehl aus:

```
python --version
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer für Python 2 kleiner als 2.7 oder für Python 3 kleiner als 3.3 ist, installieren Sie Python 2.7+ oder Python 3.3+ gemäß den Anweisungen unter [Downloading Python](#). Weitere Informationen finden Sie unter [Using Python on a Macintosh](#).

2. Führen Sie im Terminalfenster die folgenden Befehle aus, um die OpenSSL-Version zu ermitteln:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Notieren Sie sich den Wert der OpenSSL-Version.

Note

Wenn Sie Python 3 ausführen, verwenden Sie `print(ssl.OPENSSL_VERSION)`.

Um die Python-Shell zu schließen, führen Sie den folgenden Befehl aus:

```
>>>exit()
```

Wenn die OpenSSL Version 1.0.1 oder höher ist, gehen Sie direkt zu [Schritt c](#). Führen Sie andernfalls die folgenden Schritte aus:

- Führen Sie im Terminalfenster den folgenden Befehl aus, um zu festzustellen, ob der Computer das Simple Python Version Management verwendet:

```
which pyenv
```

Wird ein Dateipfad zurückgegeben, wählen Sie die Registerkarte Using **pyenv** (Unter Verwendung von). Wird nichts zurückgegeben, wählen Sie die Registerkarte Not using **pyenv** (Nicht unter Verwendung von).

Using pyenv

1. Weitere Informationen, wie Sie die neueste stabile Python-Version bestimmen, finden Sie unter [Python Releases for Max OS X](#) (oder ähnlich). Im folgenden Beispiel wird dieser Wert durch *latest-Python-version* angezeigt.
2. Führen Sie im Terminalfenster die folgenden Befehle aus:

```
pyenv install latest-Python-version  
pyenv global latest-Python-version
```

Wenn beispielsweise die neueste Version für Python 2 gleich 2.7.14 ist, lauten diese Befehle:

```
pyenv install 2.7.14  
pyenv global 2.7.14
```

3. Schließen Sie das Terminalfenster und öffnen Sie es wieder und führen Sie dann die folgenden Befehle aus:

```
python
```



```
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Die OpenSSL-Version sollte 1.0.1 oder neuer sein. Wenn die Version kleiner als 1.0.1 ist, dann ist die Aktualisierung fehlgeschlagen. Überprüfen Sie die Python-Version in den Befehlen `pyenv install` und `pyenv global` und versuchen Sie es erneut.

4. Um die Python-Shell zu schließen, führen Sie den folgenden Befehl aus:

```
exit()
```

Not using pyenv

1. Führen Sie den folgenden Befehl in einem Terminalfenster aus, um festzustellen, ob [brew](#) installiert ist:

```
which brew
```

Wird kein Dateipfad zurückgegeben, installieren Sie `brew` wie folgt:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Note

Befolgen Sie die Eingabeaufforderungen für die Installation. Der Download für die Xcode-Befehlszeilentools kann einige Zeit in Anspruch nehmen.

2. Führen Sie die folgenden Befehle aus:

```
brew update
brew install openssl
brew install python@2
```

AWS IoT Device SDK Für Python ist OpenSSL Version 1.0.1 (oder höher) erforderlich, die mit der ausführbaren Python-Datei kompiliert wurde. Der obige Befehl `brew install python` installiert eine ausführbare `python2`-Datei, die diese

Anforderung erfüllt. Die ausführbare python2-Datei wird im Ordner `/usr/local/bin` installiert, das Teil der Umgebungsvariablen PATH sein sollte. Führen Sie zur Bestätigung den folgenden Befehl aus:

```
python2 --version
```

Wenn python2-Versionsinformationen bereitgestellt werden, gehen Sie zum nächsten Schritt über. Andernfalls fügen Sie Ihrer Umgebungsvariablen PATH dauerhaft den Pfad `/usr/local/bin` hinzu, indem Sie Ihrem Shell-Profil die folgende Zeile hinzufügen:

```
export PATH="/usr/local/bin:$PATH"
```

Wenn Sie beispielsweise `.bash_profile` verwenden oder noch kein Shell-Profil haben, führen Sie den folgenden Code von einem Terminalfenster aus aus:

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

Anschließend führen Sie [source](#) für Ihr Shell-Profil aus und bestätigen, dass `python2 --version` Versionsinformationen bereitstellt. Wenn Sie z. B. `.bash_profile` verwenden, führen Sie die folgenden Befehle aus:

```
source ~/.bash_profile  
python2 --version
```

python2-Versionsinformationen sollten zurückgegeben werden.

3. Fügen Sie Ihrem Shell-Profil die folgende Zeile hinzu:

```
alias python="python2"
```

Wenn Sie beispielsweise `.bash_profile` verwenden oder noch kein Shell-Profil haben, führen Sie den folgenden Code aus:

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. Anschließend führen Sie [source](#) für Ihr Shell-Profil aus. Wenn Sie z. B. `.bash_profile` verwenden, führen Sie den folgenden Befehl aus:

```
source ~/.bash_profile
```

Wenn der Befehl `python` aufgerufen wird, wird die Python-Programmdateien ausgeführt, in der die erforderliche OpenSSL-Version (python2) enthalten ist.

5. Führen Sie die folgenden Befehle aus:

```
python
import ssl
print ssl.OPENSSL_VERSION
```

Die OpenSSL-Version sollte 1.0.1 oder neuer sein.

6. Um die Python-Shell zu schließen, führen Sie den folgenden Befehl aus:

```
exit()
```

3. Führen Sie die folgenden Befehle aus, um das AWS IoT Device SDK für Python zu installieren:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

UNIX-like system

1. Führen Sie in einem Terminalfenster den folgenden Befehl aus:

```
python --version
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer für Python 2 kleiner als 2.7 oder für Python 3 kleiner als 3.3 ist, installieren Sie Python 2.7+ oder Python 3.3+ gemäß den Anweisungen unter [Downloading Python](#). Weitere Informationen finden Sie unter [Using Python on Unix-Plattformen](#).

2. Führen Sie im Terminal die folgenden Befehle aus, um die OpenSSL-Version zu ermitteln:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Notieren Sie sich den Wert der OpenSSL-Version.

 Note

Wenn Sie Python 3 ausführen, verwenden Sie `print(ssl.OPENSSL_VERSION)`.

Um die Python-Shell zu schließen, führen Sie den folgenden Befehl aus:

```
exit()
```

Wenn die OpenSSL Version 1.0.1 oder höher ist, gehen Sie direkt zum nächsten Schritt. Andernfalls führen Sie den Befehl/die Befehle für die Aktualisierung von OpenSSL für Ihre Verteilung aus (zum Beispiel `sudo yum update openssl`, `sudo apt-get update` usw.).

Vergewissern Sie sich, dass die OpenSSL-Version 1.0.1 oder höher ist, indem Sie die folgenden Befehle ausführen:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. Führen Sie die folgenden Befehle aus, um das AWS IoT Device SDK für Python zu installieren:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. Navigieren Sie nach der Installation von AWS IoT Device SDK for Python zu dem `samples` Ordner und öffnen Sie den `greengrass` Ordner.

Für dieses Tutorial kopieren Sie die `basicDiscovery.py`-Beispielfunktion, die die Zertifikate und Schlüssel verwendet, die Sie in [the section called “Client-Geräte in einer AWS IoT Greengrass Gruppe erstellen”](#) heruntergeladen haben.

3. Kopieren Sie `basicDiscovery.py` in den Ordner, der die Gerätezertifikate und HelloWorld Schlüssel `_Publisher` und HelloWorld `_Subscriber` enthält.

Testen der Kommunikation

1. Stellen Sie sicher, dass Ihr Computer und das AWS IoT Greengrass Kerngerät über dasselbe Netzwerk mit dem Internet verbunden sind.
 - a. Führen Sie auf dem AWS IoT Greengrass Core-Gerät den folgenden Befehl aus, um dessen IP-Adresse zu ermitteln.

```
hostname -I
```

- b. Führen Sie auf Ihrem Computer mit der IP-Adresse des Cores den folgenden Befehl aus. Mit Ctrl + C können Sie den Befehl ping stoppen.

```
ping IP-address
```

Eine Ausgabe, die der folgenden ähnelt, weist auf eine erfolgreiche Kommunikation zwischen dem Computer und dem AWS IoT Greengrass Kerngerät hin (0% Paketverlust):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

 Note

Wenn Sie eine laufende EC2 Instance nicht pingen können, stellen Sie sicher AWS IoT Greengrass, dass die Regeln für eingehende Sicherheitsgruppen für die Instance ICMP-Verkehr für [Echo-Anforderungsnachrichten](#) zulassen. Weitere Informationen finden Sie unter [Regeln zu einer Sicherheitsgruppe hinzufügen](#) im EC2 Amazon-Benutzerhandbuch.

Auf Windows-Hostcomputern müssen Sie in der App „Windows-Firewall mit erweiterter Sicherheit“ möglicherweise auch eine Regel für eingehende Echoanfragen aktivieren (z. B. Datei- und Druckerfreigabe (Echo Request — ICMPv4 -In)), oder eine Regel erstellen.

2. Holen Sie sich Ihren Endpunkt AWS IoT .

- a. Wählen Sie im Navigationsbereich der [AWS IoT Konsole](#) Einstellungen aus.
- b. Notieren Sie sich unter Endpunkt für Gerätedaten den Wert von Endpoint. Sie verwenden diesen Wert, um den `AWS_IOT_ENDPOINT` Platzhalter in den Befehlen in den folgenden Schritten zu ersetzen.

 Note

Stellen Sie sicher, dass Ihre [Endpunkte Ihrem Zertifikatstyp entsprechen](#).

3. Öffnen Sie auf Ihrem Computer (nicht auf dem AWS IoT Greengrass Hauptgerät) zwei [Befehlszeilenfenster](#) (Terminal oder Eingabeaufforderung). Ein Fenster steht für das HelloWorld _Publisher-Clientgerät und das andere für das HelloWorld _Subscriber-Clientgerät.

`basicDiscovery.py` Versucht bei der Ausführung, Informationen über den Standort des AWS IoT Greengrass Kerns an seinen Endpunkten zu sammeln. Diese Informationen werden gespeichert, nachdem das Client-Gerät den Core erkannt und erfolgreich eine Verbindung mit ihm hergestellt hat. Auf diese Weise ist künftig ein lokales Ausführen (ohne Internetverbindung) von Messaging-Funktionen und Operationen möglich.

 Note

Der für MQTT-Verbindungen IDs verwendete Client muss mit dem Ding-Namen des Client-Geräts übereinstimmen. Das `basicDiscovery.py` Skript setzt die Client-ID für

MQTT-Verbindungen auf den Namen des Dings, den Sie bei der Ausführung des Skripts angeben.

Führen Sie den folgenden Befehl in dem Ordner aus, der die `basicDiscovery.py` Datei enthält, um detaillierte Informationen zur Verwendung des Skripts zu erhalten:

```
python basicDiscovery.py --help
```

4. Führen Sie im Fenster des HelloWorld_Publisher-Client-Geräts die folgenden Befehle aus.

- *path-to-certs-folder* Ersetzen Sie es durch den Pfad zu dem Ordner, der die Zertifikate, Schlüssel und `basicDiscovery.py` enthält.
- Ersetzen Sie *AWS_IOT_ENDPOINT* durch Ihren Endpunkt.
- Ersetzen Sie die beiden *publisherCertId* Instanzen durch die Zertifikat-ID im Dateinamen Ihres HelloWorld_Publisher-Client-Geräts.

```
cd path-to-certs-folder  
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem  
--cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key  
--thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --  
message 'Hello, World! Sent from HelloWorld_Publisher'
```

Die Ausgabe sollte ungefähr wie die folgende aussehen und kann beispielsweise die nachstehenden Einträge umfassen: Published topic 'hello/world/pubsub':
{"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}.

Note

Wenn das Skript eine `error: unrecognized arguments`-Nachricht zurückgibt, ändern Sie die einfachen Anführungszeichen zu doppelten Anführungszeichen für die Parameter `--topic` und `--message` und führen Sie den Befehl erneut aus. Um einen Verbindungsfehler zu beheben, können Sie versuchen, das Problem mithilfe der [manuellen IP-Erkennung](#) zu lösen.


```
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
2017-11-13 21:12:26,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:26,297 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:27,301 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
2017-11-13 21:12:27,302 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:27,303 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:28,305 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
2017-11-13 21:12:28,306 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:28,307 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:29,310 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
```

5. Führen Sie im Fenster des HelloWorld_Subscriber-Client-Geräts die folgenden Befehle aus.

- *path-to-certs-folder* Ersetzen Sie es durch den Pfad zu dem Ordner, der die Zertifikate, Schlüssel und `basicDiscovery.py` enthält.
- Ersetzen Sie *AWS_IOT_ENDPOINT* durch Ihren Endpunkt.
- Ersetzen Sie die beiden *subscriberCertId* Instanzen durch die Zertifikat-ID im Dateinamen Ihres HelloWorld_Subscriber-Client-Geräts.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
```

Die Ausgabe sollte folgendermaßen aussehen und kann beispielsweise nachstehende Einträge umfassen: Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}.

```
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:27,436 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
2017-11-13 21:12:28,320 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
2017-11-13 21:12:29,547 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
```


Schließen Sie den HelloWorld_Publisher Fenster, um zu verhindern, dass Nachrichten im HelloWorld_Subscriber Fenster.

Tests in einem Unternehmensnetzwerk können die Verbindung zum Core beeinträchtigen. Um dieses Problem zu umgehen, können Sie den Endpunkt manuell eingeben. Dadurch wird sichergestellt, dass das `basicDiscovery.py` Skript eine Verbindung zur richtigen IP-Adresse des AWS IoT Greengrass Kerngeräts herstellt.

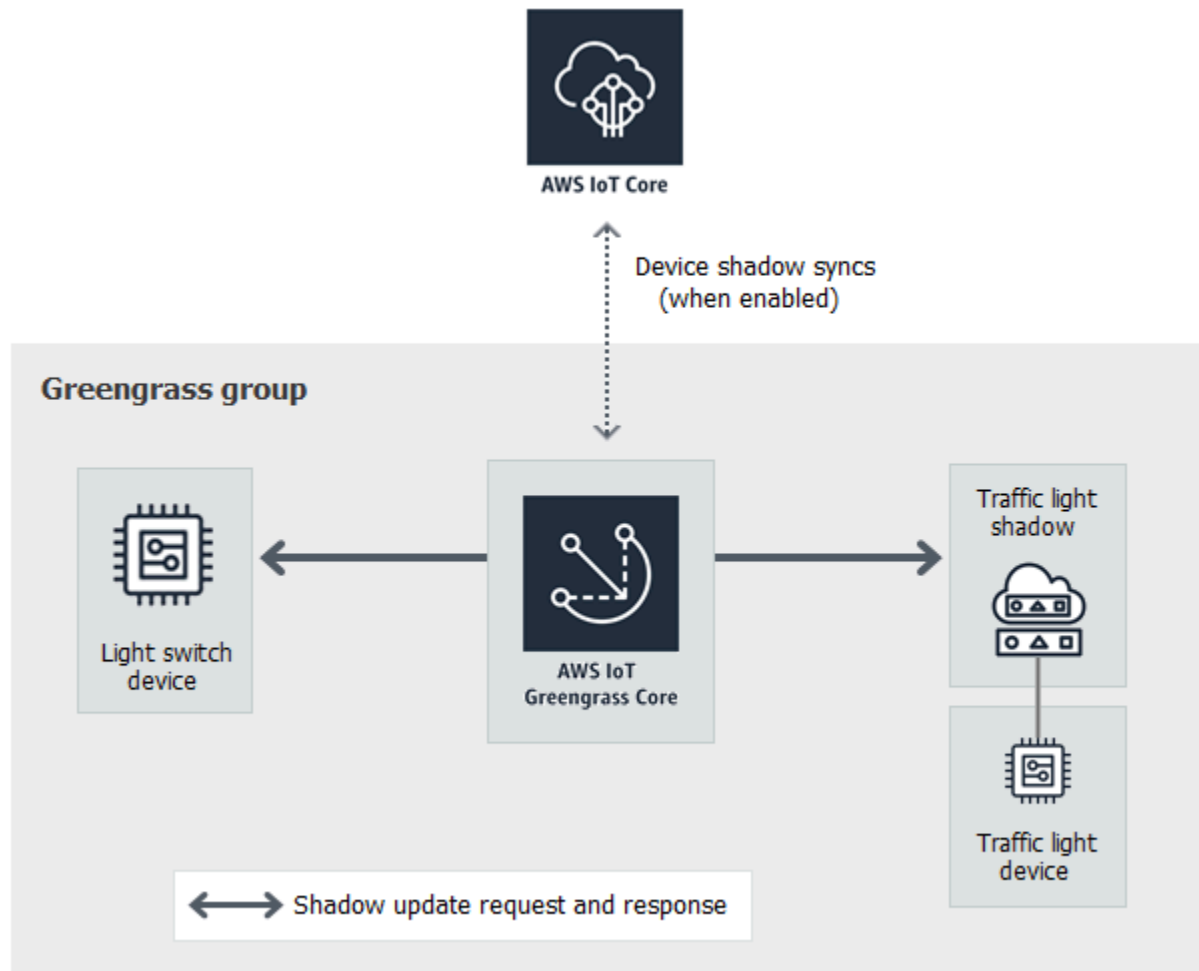
Endpunkt manuell eingeben

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie unter Greengrass-Gruppen Ihre Gruppe aus.
3. Konfigurieren Sie den Core für die manuelle Verwaltung von MQTT-Broker-Endpunkten. Gehen Sie wie folgt vor:
 - a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
 - b. Wählen Sie unter System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
 - c. Wählen Sie unter IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte manuell verwalten und dann Speichern aus.
4. Geben Sie den MQTT-Broker-Endpunkt für den Core ein. Gehen Sie wie folgt vor:
 - a. Wählen Sie unter Übersicht den Greengrass-Kern aus.
 - b. Wählen Sie unter MQTT-Broker-Endpunkte die Option Endpunkte verwalten aus.
 - c. Wählen Sie Endpunkt hinzufügen und stellen Sie sicher, dass Sie nur einen Endpunktwert haben. Dieser Wert muss der Endpunkt der IP-Adresse für Port 8883 Ihres AWS IoT Greengrass Kerngeräts sein (z. B. 192.168.1.4).
 - d. Wählen Sie Aktualisieren.

Modul 5: Interaktion mit Geräteschatten

Dieses erweiterte Modul zeigt Ihnen, wie Client-Geräte mit [AWS IoT Geräteschatten](#) in einer AWS IoT Greengrass Gruppe interagieren können. Ein Schatten ist ein JSON-Dokument, das für die Speicherung der aktuellen oder gewünschten Statusinformationen für einen Gegenstand verwendet wird. In diesem Modul erfahren Sie, wie ein Client-Gerät (GG_Switch) den Status eines

anderen Client-Geräts (GG_TrafficLight) ändern kann und wie diese Zustände mit der AWS IoT Greengrass Cloud synchronisiert werden können:



Bevor Sie beginnen, führen Sie das [Greengrass Device Setup](#)-Skript aus oder stellen Sie sicher, dass Sie [Modul 1](#) und [Modul 2](#) abgeschlossen haben. Sie sollten auch wissen, wie Client-Geräte mit einem AWS IoT Greengrass Core verbunden werden ([Modul 4](#)). Sie benötigen keine anderen Komponenten oder Geräte.

Dieses Modul sollte etwa 30 Minuten in Anspruch nehmen.

Themen

- [Konfigurieren von Geräten und Abonnements](#)
- [Download der benötigten Dateien](#)
- [Testen der Kommunikation \(Gerätesynchronisierungen deaktiviert\)](#)
- [Testen der Kommunikation \(Gerätesynchronisierungen aktiviert\)](#)

Konfigurieren von Geräten und Abonnements

Schatten können synchronisiert werden, AWS IoT wenn der AWS IoT Greengrass Core mit dem Internet verbunden ist. In diesem Modul verwenden Sie zunächst die lokalen Schattengeräte ohne ein Synchronisieren mit der Cloud. Anschließend aktivieren Sie die Cloud-Synchronisierung.

Jedes Client-Gerät hat seinen eigenen Schatten. Weitere Informationen finden Sie unter [Device Shadow Service for AWS IoT](#) im AWS IoT Entwicklerhandbuch.

1. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Registerkarte Client-Geräte aus.
2. Fügen Sie auf der Registerkarte Client-Geräte zwei neue Client-Geräte zu Ihrer AWS IoT Greengrass Gruppe hinzu. Ausführliche Informationen zu den Schritten dieses Prozesses finden Sie unter [the section called “Client-Geräte in einer AWS IoT Greengrass Gruppe erstellen”](#).
 - Benennen Sie die Client-Geräte **GG_Switch** und **GG_TrafficLight**.
 - Generieren Sie die Sicherheitsressourcen für beide Client-Geräte und laden Sie sie herunter.
 - Notieren Sie sich die Zertifikat-ID in den Dateinamen der Sicherheitsressourcen für die Client-Geräte. Sie werden diese Werte in einem späteren Schritt verwenden.
3. Erstellen Sie auf Ihrem Computer einen Ordner für die Sicherheitsanmeldeinformationen dieser Client-Geräte. Kopieren Sie die Zertifikate und Schlüssel in diesen Ordner.
4. Stellen Sie sicher, dass die Client-Geräte so eingestellt sind, dass sie lokale Schatten verwenden und nicht mit dem synchronisieren AWS Cloud. Wenn nicht, wählen Sie das Client-Gerät aus, wählen Sie Shadow synchronisieren und anschließend Shadow-Synchronisierung mit Cloud deaktivieren.
5. Fügen Sie die Abonnements der folgenden Tabelle zu Ihrer Gruppe hinzu. So erstellen Sie beispielsweise das erste Abonnement:
 - a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Hinzufügen aus.
 - b. Wählen Sie als Quelltyp die Option Client-Gerät und dann GG_Switch aus.
 - c. Wählen Sie als Zieltyp Service und dann Local Shadow Service aus.
 - d. Geben Sie unter Topic filter (Themenfilter) **\$aws/things/GG_TrafficLight/shadow/update** ein.
 - e. Wählen Sie Create subscription (Abonnement erstellen) aus.

Die Themen müssen genau wie in der Tabelle eingegeben werden. Sie können zwar Platzhalter verwenden, um einige der Abonnements zu konsolidieren, dies wird jedoch nicht empfohlen. Weitere Informationen finden Sie unter [Shadow MQTT-Themen](#) im AWS IoT Developer Guide.

Quelle	Ziel	Thema	Hinweise
GG_Switch	Local Shadow Service	\$aws/things/GG_TrafficLight/shadow/update	Der GG_Switch sendet eine Aktualisierungsanfrage an das Aktualisierungsthema.
Local Shadow Service	GG_Switch	\$aws/things/GG_TrafficLight/shadow/update/accepted	Die GG_Switch muss wissen, ob die Aktualisierungsanforderung akzeptiert wurde.
Local Shadow Service	GG_Switch	\$aws/things/GG_TrafficLight/shadow/update/rejected	Die GG_Switch muss wissen, ob die Aktualisierungsanforderung abgelehnt wurde.
GG_TrafficLight	Local Shadow Service	\$aws/things/GG_TrafficLight/shadow/update	GG_TrafficLight sendet eine Aktualisierung seines Status an das Aktualisierungsthema.
Local Shadow Service	GG_TrafficLight	\$aws/things/GG_TrafficLight/shadow/update/delta	Der Local Shadow Service sendet ein empfangenes Update TrafficLight über das Delta-Thema an GG_.

Quelle	Ziel	Thema	Hinweise
Local Shadow Service	GG_TrafficLight	\$saws/things/GG_TrafficLight/shadow/update/accepted	GG_TrafficLight muss wissen, ob sein Status-Update akzeptiert wurde.
Local Shadow Service	GG_TrafficLight	\$saws/things/GG_TrafficLight/shadow/update/rejected	GG_TrafficLight muss wissen, ob sein Status-Update abgelehnt wurde.

Die neuen Abonnements werden auf der Registerkarte Abonnements angezeigt.

Note

Weitere Informationen zum \$-Zeichen finden Sie im Abschnitt über [Reservierte Themen](#).

6. Stellen Sie sicher, dass die automatische Erkennung aktiviert ist, damit der Greengrass Core eine Liste seiner IP-Adressen veröffentlichen kann. Client-Geräte verwenden diese Informationen, um den Core zu ermitteln. Gehen Sie wie folgt vor:
 - a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
 - b. Wählen Sie unter System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
 - c. Wählen Sie unter IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben aus und wählen Sie dann Speichern aus.
7. Stellen Sie sicher, dass der Greengrass-Daemon läuft, wie unter beschrieben. [Bereitstellen von Cloud-Konfigurationen für ein Core-Gerät](#)
8. Wählen Sie auf der Gruppenkonfigurationsseite Deploy aus.

Download der benötigten Dateien

1. Falls Sie dies noch nicht getan haben, installieren Sie das AWS IoT Device SDK für Python. Anweisungen hierzu finden Sie in Schritt 1 unter [the section called "Installieren Sie das AWS IoT Device SDK für Python"](#).

Dieses SDK wird von Client-Geräten für die Kommunikation mit AWS IoT und mit AWS IoT Greengrass Kerngeräten verwendet.

2. Laden Sie aus dem Ordner mit den [TrafficLight](#) Beispielen die `trafficLight.py` Dateien `lightController.py` und auf GitHub Ihren Computer herunter. Speichern Sie sie in dem Ordner, der die Zertifikate und Schlüssel der TrafficLight Client-Geräte `GG_Switch` und `GG_` enthält.

Das `lightController.py` Skript entspricht dem `GG_Switch`-Clientgerät und das `trafficLight.py` Skript entspricht dem `GG_`-Clientgerät. TrafficLight



7aa87aa1cf.cert.pem
7aa87aa1cf.private.key
7aa87aa1cf.public.key
a27b261ea9.cert.pem
a27b261ea9.private.key
a27b261ea9.public.key
lightController.py
root-ca-cert.pem
trafficLight.py

Note

Die Python-Beispieldateien werden der Einfachheit halber im AWS IoT Greengrass Core SDK for Python-Repository gespeichert, sie verwenden jedoch nicht das AWS IoT Greengrass Core SDK.

Testen der Kommunikation (Gerätesynchronisierungen deaktiviert)

1. Stellen Sie sicher, dass Ihr Computer und das AWS IoT Greengrass Kerngerät über dasselbe Netzwerk mit dem Internet verbunden sind.
 - a. Führen Sie auf dem AWS IoT Greengrass Core-Gerät den folgenden Befehl aus, um dessen IP-Adresse zu ermitteln.

```
hostname -I
```

- b. Führen Sie auf Ihrem Computer mit der IP-Adresse des Cores den folgenden Befehl aus. Mit Ctrl + C können Sie den Befehl ping stoppen.

```
ping IP-address
```

Eine Ausgabe, die der folgenden ähnelt, weist auf eine erfolgreiche Kommunikation zwischen dem Computer und dem AWS IoT Greengrass Kerngerät hin (0% Paketverlust):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

Note

Wenn Sie eine laufende EC2 Instance nicht pingen können, stellen Sie sicher AWS IoT Greengrass, dass die Regeln für eingehende Sicherheitsgruppen für die Instance ICMP-Verkehr für [Echo-Anforderungsnachrichten](#) zulassen. Weitere Informationen finden Sie unter [Regeln zu einer Sicherheitsgruppe hinzufügen](#) im EC2 Amazon-Benutzerhandbuch.

Auf Windows-Hostcomputern müssen Sie in der App „Windows-Firewall mit erweiterter Sicherheit“ möglicherweise auch eine Regel für eingehende Echoanfragen aktivieren (z. B. Datei- und Druckerfreigabe (Echo Request — ICMPv4 -In)), oder eine Regel erstellen.

2. Holen Sie sich Ihren Endpunkt AWS IoT .
 - a. Wählen Sie im Navigationsbereich der [AWS IoT Konsole](#) Einstellungen aus.
 - b. Notieren Sie sich unter Endpunkt für Gerätedaten den Wert von Endpoint. Sie verwenden diesen Wert, um den `AWS_IOT_ENDPOINT` Platzhalter in den Befehlen in den folgenden Schritten zu ersetzen.

 Note

Stellen Sie sicher, dass Ihre [Endpunkte Ihrem Zertifikatstyp entsprechen](#).

3. Öffnen Sie auf Ihrem Computer (nicht auf dem AWS IoT Greengrass Hauptgerät) zwei [Befehlszeilenfenster](#) (Terminal oder Eingabeaufforderung). Ein Fenster steht für das GG_Switch-Client-Gerät und das andere für das GG_Client-Gerät. TrafficLight

a. Führen Sie im Fenster des GG_Switch-Client-Geräts die folgenden Befehle aus.

- *path-to-certs-folder* Ersetzen Sie durch den Pfad zu dem Ordner, der die Zertifikate, Schlüssel und Python-Dateien enthält.
- Ersetzen Sie *AWS_IOT_ENDPOINT* durch Ihren Endpunkt.
- Ersetzen Sie die beiden *switchCertId* Instanzen durch die Zertifikat-ID im Dateinamen Ihres GG_Switch-Client-Geräts.

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA
  AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-
  private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
```

b. Führen Sie im Fenster des TrafficLight GG_Client-Geräts die folgenden Befehle aus.

- *path-to-certs-folder* Ersetzen Sie durch den Pfad zu dem Ordner, der die Zertifikate, Schlüssel und Python-Dateien enthält.
- Ersetzen Sie *AWS_IOT_ENDPOINT* durch Ihren Endpunkt.
- Ersetzen Sie die beiden *lightCertId* Instanzen durch die Zertifikat-ID im Dateinamen Ihres TrafficLight GG_Client-Geräts.

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
  --cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
  thingName GG_TrafficLight --clientId GG_TrafficLight
```

Alle 20 Sekunden aktualisiert der Schalter den Schattenzustand mit „G“, „Y“ und „R“ und die Lampe zeigt ihren neuen Zustand an, wie nachfolgend dargestellt.

GG_Switch Ausgabe:

```
{ "state": { "desired": { "property": "R" } } }
2018-12-20 12:23:01,446 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~Shadow Update Accepted~~~~~
Update request with token: 3b22e27c-930d-4c6a-8562-9f86088249f4 accepted!
property: R
~~~~~
```

TrafficLight GG_-Ausgabe:

```
+++++++ Received Shadow Delta ++++++++
{ 'state': { 'u'property': u'R'}, 'u'metadata': { 'u'property': { 'u'timestamp': 1545337381 }, 'u'version': 33, 'u'clientToken':
u'3b22e27c-930d-4c6a-8562-9f86088249f4' } }
property: R
version: 33
+++++++

Light changed to: R
{ "state": { "reported": { "property": "R" } } }
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~Shadow Update Accepted~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~
```

Bei der ersten Ausführung führt jedes Client-Geräteskript den AWS IoT Greengrass Erkennungsdienst aus, um eine Verbindung zum AWS IoT Greengrass Core herzustellen (über das Internet). Nachdem ein Client-Gerät den AWS IoT Greengrass Core erkannt und erfolgreich mit ihm verbunden hat, können future Operationen lokal ausgeführt werden.

 Note

Die Skripts `trafficLight.py` und `lightController.py` speichern Verbindungsinformationen im Ordner `groupCA`. Dieser wird im selben Ordner wie die Skripts erstellt. Wenn Sie Verbindungsfehler erhalten, stellen Sie sicher, dass die IP-Adresse in der `ggc-host` Datei mit dem IP-Adressendpunkt für Ihren Core übereinstimmt.

4. Wählen Sie in der AWS IoT Konsole Ihre AWS IoT Greengrass Gruppe aus, klicken Sie auf die Registerkarte Client-Geräte und dann auf GG_, TrafficLight um die Seite mit den Ding-Details des AWS IoT Client-Geräts zu öffnen.
5. Wählen Sie den Tab „Geräteschatten“. Nachdem der GG_Switch den Status geändert hat, sollte es keine Updates für diesen Shadow geben. Das liegt daran, dass GG_ auf „Schattensynchronisierung mit der Cloud deaktivieren“ eingestellt TrafficLight ist.

6. Drücken Sie `Ctrl + C` im Fenster des `GG_Switch` (`lightController.py`)-Client-Geräts. Sie sollten sehen, dass das `GG_TrafficLight` (`trafficLight.py`)-Fenster keine Statusänderungsnachrichten mehr empfängt.

Schließen Sie diese Fenster nicht, damit Sie die Befehle im nächsten Abschnitt erneut ausführen können.

Testen der Kommunikation (Gerätesynchronisierungen aktiviert)

Für diesen Test konfigurieren Sie den `GG_TrafficLight` Device Shadow, mit dem synchronisiert werden soll AWS IoT. Führen Sie die gleichen Befehle wie im vorherigen Test aus, aber dieses Mal wird der Schattenzustand in der Cloud aktualisiert, sobald `GG_Switch` eine Aktualisierungsanforderung sendet.

1. Wählen Sie in der AWS IoT Konsole Ihre AWS IoT Greengrass Gruppe und dann die Registerkarte Client-Geräte aus.
2. Wählen Sie das `TrafficLight` `GG_-Gerät` aus, wählen Sie Shadow synchronisieren und anschließend Shadow-Synchronisierung mit Cloud aktivieren aus.

Sie sollten eine Benachrichtigung erhalten, dass der Shadow-Sync-Status des Geräts aktualisiert wurde.

3. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Bereitstellen aus.
4. [Führen Sie in Ihren beiden Befehlszeilenfenstern die Befehle aus dem vorherigen Test für die Client-Geräte `GG\_Switch` und `GG\_ aus. TrafficLight`](#)
5. Überprüfen Sie nun den Shadow-Status in der Konsole. AWS IoT Wählen Sie Ihre AWS IoT Greengrass Gruppe aus, wählen Sie die Registerkarte Client-Geräte, wählen Sie `GG_TrafficLight`, wählen Sie die Registerkarte Geräteschatten und dann Classic Shadow.

Da Sie die Synchronisierung von `GG_TrafficLight` Shadow mit aktiviert haben AWS IoT, sollte der Shadow-Status in der Cloud jedes Mal aktualisiert werden, wenn `GG_Switch` ein Update sendet. Diese Funktionalität kann verwendet werden, um den Status eines Client-Geräts offenzulegen. AWS IoT

Note

Falls erforderlich, können Sie Probleme beheben, indem Sie sich die AWS IoT Greengrass wichtigsten Protokolle ansehen, insbesondere `runtime.log`:

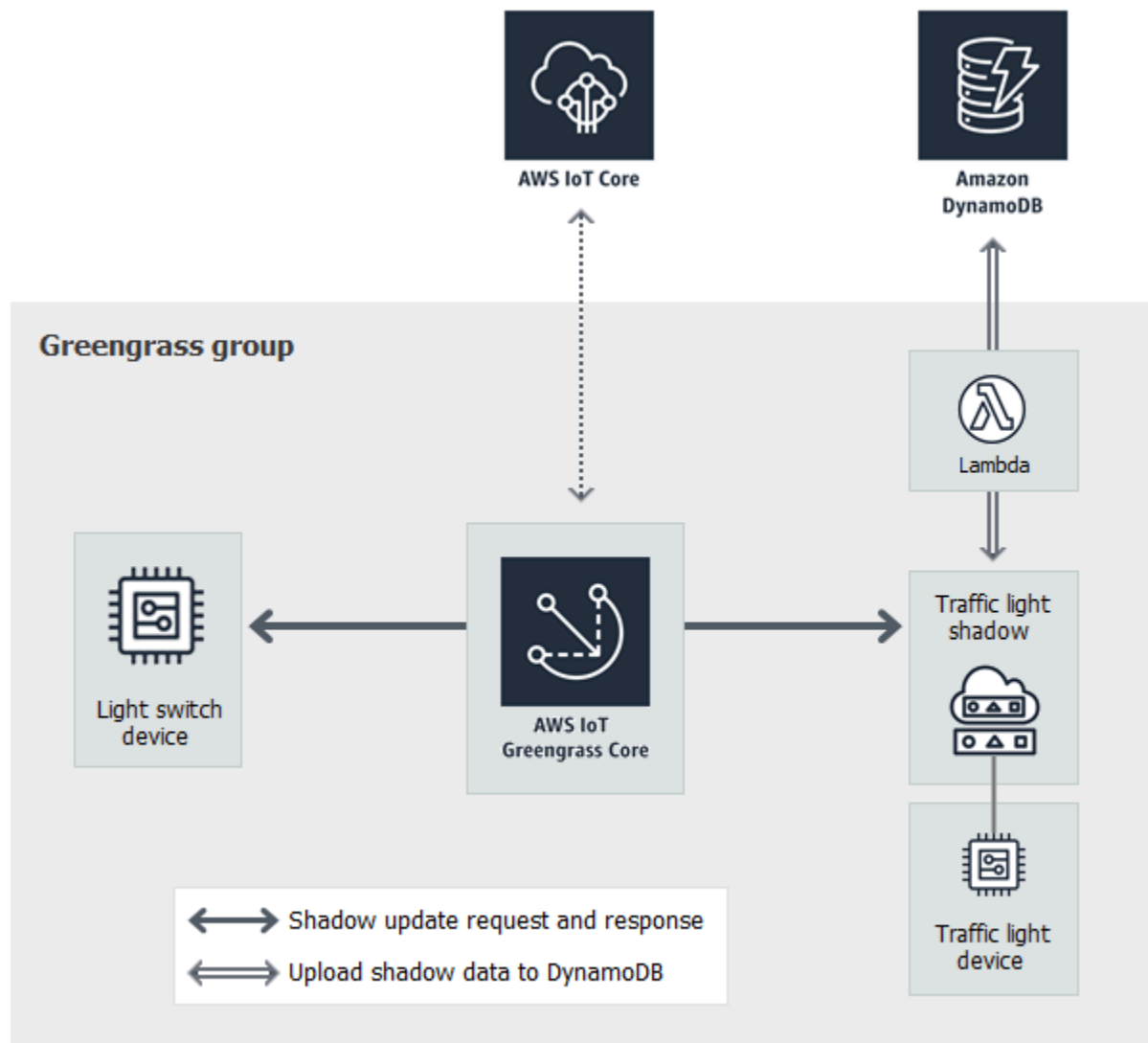
```
cd /greengrass/ggc/var/log  
sudo cat system/runtime.log | more
```

Sie können auch `GGShadowSyncManager.log` und `GGShadowService.log` einsehen. Weitere Informationen finden Sie unter [Fehlerbehebung](#).

Halten Sie die Client-Geräte und Abonnements eingerichtet. Sie werden sie im nächsten Modul wieder verwenden. Zudem werden Sie die gleichen Befehle ausführen.

Modul 6: Zugriff auf andere AWS Dienste

Dieses Modul für Fortgeschrittene zeigt Ihnen, wie AWS IoT Greengrass Kerne mit anderen AWS Diensten in der Cloud interagieren können. Es baut auf dem Ampelbeispiel aus [Modul 5](#) auf und fügt eine Lambda-Funktion hinzu, die Schattenzustände verarbeitet und eine Zusammenfassung in eine Amazon DynamoDB-Tabelle hochlädt.



Bevor Sie beginnen, führen Sie das [Greengrass Device Setup](#)-Skript aus oder stellen Sie sicher, dass Sie [Modul 1](#) und [Modul 2](#) abgeschlossen haben. Sie sollten auch [Modul 5](#) abschließen. Sie benötigen keine anderen Komponenten oder Geräte.

Dieses Modul sollte etwa 30 Minuten in Anspruch nehmen.

Note

Dieses Modul erstellt und aktualisiert eine Tabelle in DynamoDB. Obwohl die meisten Operationen klein sind und unter das kostenlose Kontingent von Amazon Web Services fallen, kann die Ausführung einiger Schritte in diesem Modul dazu führen, dass Ihr

Konto belastet wird. Informationen zur Preisgestaltung finden Sie in der [DynamoDB-Preisdokumentation](#).

Themen

- [Konfigurieren der Gruppenrolle](#)
- [Die Lambda-Funktion erstellen und konfigurieren](#)
- [Konfigurieren von Abonnements](#)
- [Testen der Kommunikation](#)

Konfigurieren der Gruppenrolle

Die Gruppenrolle ist eine [IAM-Rolle](#), die Sie erstellen und Ihrer Greengrass-Gruppe zuordnen. Diese Rolle enthält die Berechtigungen, die bereitgestellte Lambda-Funktionen (und andere AWS IoT Greengrass Funktionen) für den Zugriff auf AWS Dienste verwenden. Weitere Informationen finden Sie unter [the section called “Greengrass-Gruppenrolle.”](#).

Sie verwenden die folgenden allgemeinen Schritte, um eine Gruppenrolle in der IAM-Konsole zu erstellen.

1. Erstellen Sie eine Richtlinie, die Aktionen für eine oder mehrere Ressourcen zulässt oder verweigert.
2. Erstellen Sie eine Rolle, die den Greengrass-Service als vertrauenswürdige Entität verwendet.
3. Hängen Sie Ihre Richtlinie an die Rolle an.

Anschließend fügen Sie die Rolle in der AWS IoT Konsole der Greengrass-Gruppe hinzu.

Note

Eine Greengrass-Gruppe verfügt über eine Gruppenrolle. Wenn Sie Berechtigungen hinzufügen möchten, können Sie angehängte Richtlinien bearbeiten oder weitere Richtlinien anhängen.

Für dieses Tutorial erstellen Sie eine Berechtigungsrichtlinie, die das Beschreiben, Erstellen und Aktualisieren von Aktionen in einer Amazon DynamoDB-Tabelle ermöglicht. Anschließend fügen Sie die Richtlinie an eine neue Rolle an und ordnen die Rolle Ihrer Greengrass-Gruppe zu.

Erstellen Sie zunächst eine vom Kunden verwaltete Richtlinie, die die für die Lambda-Funktion in diesem Modul erforderlichen Berechtigungen gewährt.

1. Wählen Sie in der IAM-Konsole im Navigationsbereich Richtlinien und dann Richtlinie erstellen aus.
2. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie. Die Lambda-Funktion in diesem Modul verwendet diese Berechtigungen, um eine DynamoDB-Tabelle mit dem Namen zu erstellen und zu aktualisieren. `CarStats`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermissionsForModule6",
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable",
        "dynamodb:CreateTable",
        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
    }
  ]
}
```

3. Wählen Sie Weiter: Tags und danach Weiter: Prüfen aus. In diesem Tutorial werden keine Tags verwendet.
4. Geben Sie unter Name den Namen **greengrass_CarStats_Table** ein und wählen Sie dann Create policy (Richtlinie erstellen) aus.

Erstellen Sie als Nächstes eine Rolle, die die neue Richtlinie verwendet.

5. Wählen Sie im Navigationsbereich Rollen und dann Rolle erstellen.

6. Wählen Sie unter Typ der vertrauenswürdigen Entität die Option AWS -Service aus.
7. Wählen Sie unter Anwendungsfall, Anwendungsfälle für andere AWS Dienste die Option Greengrass, wählen Sie Greengrass und dann Weiter aus.
8. Wählen Sie unter Berechtigungsrichtlinien die neue **greengrass_CarStats_Table** Richtlinie aus und klicken Sie dann auf Weiter.
9. Geben Sie für Rollenname den Namen **Greengrass_Group_Role** ein.
10. Geben Sie für Beschreibung den Text **Greengrass group role for connectors and user-defined Lambda functions** ein.
11. Wählen Sie Rolle erstellen aus.

Fügen Sie die Rolle nun Ihrer Greengrass-Gruppe hinzu.

12. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
13. Wählen Sie unter Greengrass-Gruppen Ihre Gruppe aus.
14. Wählen Sie Einstellungen und dann Rolle als Mitarbeiter aus.
15. Wählen Sie eine Rolle Greengrass_Group_Role aus Ihrer Rollenliste und wählen Sie dann Associate-Rolle aus.

Die Lambda-Funktion erstellen und konfigurieren

In diesem Schritt erstellen Sie eine Lambda-Funktion, die die Anzahl der Autos verfolgt, die die Ampel passieren. Jedes Mal, wenn sich der GG_TrafficLight Schattenzustand ändert, simuliert die Lambda-Funktion das Vorbeifahren einer zufälligen Anzahl von Autos (von 1 bis 20). Bei jeder dritten G Lichtänderung sendet die Lambda-Funktion grundlegende Statistiken wie Min und Max an eine DynamoDB-Tabelle.

1. Erstellen Sie auf Ihrem Computer den Ordner `car_aggregator`.
2. Laden Sie die Datei aus dem GitHub Ordner mit den [TrafficLight](#) Beispielen in den `carAggregator.py` Ordner herunter. `car_aggregator` Dies ist Ihr Lambda-Funktionscode.

Note

Diese Python-Beispieldatei wird der Einfachheit halber im AWS IoT Greengrass Core SDK-Repository gespeichert, verwendet jedoch nicht das AWS IoT Greengrass Core SDK.

3. Wenn Sie nicht in der Region USA Ost (Nord-Virginia) arbeiten, öffnen Sie die Region `carAggregator.py` und wechseln Sie `region_name` in der folgenden Zeile zu der AWS-Region, die aktuell in der AWS IoT Konsole ausgewählt ist. Eine Liste der unterstützten AWS-Regionen finden Sie [AWS IoT Greengrass](#) in der Allgemeine Amazon Web Services-Referenz.

```
dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
```

4. Führen Sie den folgenden Befehl in einem [Befehlszeilenfenster](#) aus, um das [AWS SDK für Python \(Boto3\)](#) Paket und seine Abhängigkeiten im `car_aggregator` Ordner zu installieren. Die Funktionen von Greengrass Lambda verwenden das AWS SDK, um auf andere AWS Dienste zuzugreifen. (Verwenden Sie unter Windows eine [Eingabeaufforderung mit erhöhten Rechten](#)).

```
pip install boto3 -t path-to-car_aggregator-folder
```

Dadurch wird eine Verzeichnisliste ähnlich der folgenden angezeigt:

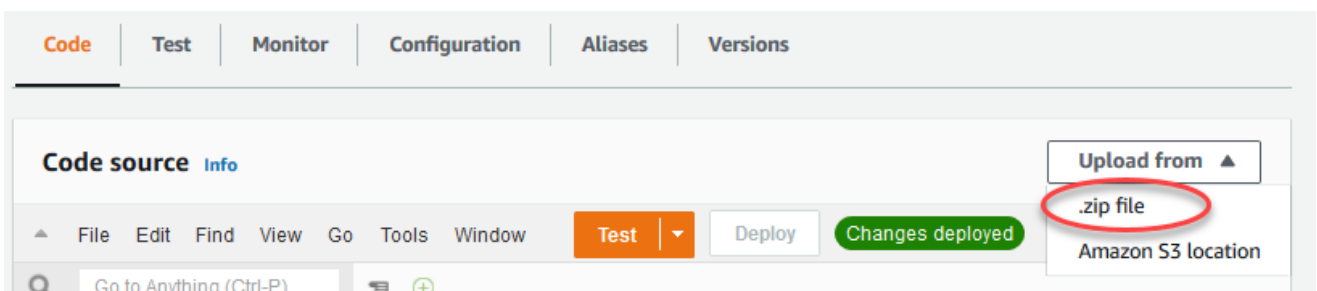
Name	Date modified	Type
 <code>bin</code>	12/31/2018 2:27 PM	File folder
 <code>boto3</code>	12/31/2018 2:27 PM	File folder
 <code>boto3-1.9.71.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>botocore</code>	12/31/2018 2:27 PM	File folder
 <code>botocore-1.12.71.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>concurrent</code>	12/31/2018 2:27 PM	File folder
 <code>dateutil</code>	12/31/2018 2:27 PM	File folder
 <code>docutils</code>	12/31/2018 2:27 PM	File folder
 <code>docutils-0.14.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>futures-3.2.0.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>jmespath</code>	12/31/2018 2:27 PM	File folder
 <code>jmespath-0.9.3.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>python_dateutil-2.7.5.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>s3transfer</code>	12/31/2018 2:27 PM	File folder
 <code>s3transfer-0.1.13.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>six-1.12.0.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>urllib3</code>	12/31/2018 2:27 PM	File folder
 <code>urllib3-1.24.1.dist-info</code>	12/31/2018 2:27 PM	File folder
 <code>carAggregator.py</code>	12/31/2018 2:25 PM	PY File
 <code>six.py</code>	12/31/2018 2:27 PM	PY File
 <code>six.pyc</code>	12/31/2018 2:27 PM	Compiled Python ...

5. Komprimieren Sie den Inhalt des Ordners `car_aggregator` in eine `.zip`-Datei namens `car_aggregator.zip`. (Komprimieren Sie den Inhalt des Ordners, nicht den Ordner.) Dies ist Ihr Lambda-Funktionsbereitstellungspaket.
6. Erstellen Sie in der Lambda-Konsole eine Funktion mit dem Namen **GG_Car_Aggregator** und legen Sie die verbleibenden Felder wie folgt fest:
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.

Wählen Sie Funktion erstellen.

The screenshot shows the 'Basic information' tab in the AWS Lambda console. The 'Function name' field is set to 'GG_Car_Aggregator'. The 'Runtime' dropdown is set to 'Python 3.7'. The 'Permissions' section shows the default execution role. At the bottom right, there are 'Cancel' and 'Create function' buttons, with the latter highlighted by a red rectangle.

7. Laden Sie das Bereitstellungspaket Ihrer Lambda-Funktion hoch:
 - a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Upload und anschließend Ihr `car_aggregator.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
 - c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Geben Sie unter Handler **`carAggregator.function_handler`** ein.
 - d. Wählen Sie Speichern.
8. Veröffentlichen Sie die Lambda-Funktion und erstellen Sie dann einen Alias mit dem Namen **GG_CarAggregator**. step-by-stepAnweisungen finden Sie in den Schritten zum [Veröffentlichen der Lambda-Funktion](#) und zum [Erstellen eines Alias](#) in Modul 3 (Teil 1).
9. Fügen Sie in der AWS IoT Konsole die Lambda-Funktion, die Sie gerade erstellt haben, zu Ihrer AWS IoT Greengrass Gruppe hinzu:
 - a. Wählen Sie auf der Gruppenkonfigurationsseite Lambda-Funktionen und dann unter Meine Lambda-Funktionen die Option Hinzufügen aus.
 - b. Wählen Sie für die Lambda-Funktion GG_Car_Aggregator.
 - c. Wählen Sie für die Lambda-Funktionsversion den Alias für die Version aus, die Sie veröffentlicht haben.
 - d. Geben Sie in Memory Limit (Speicherlimit) **64 MB** ein.
 - e. Wählen Sie für Pinned die Option True aus.
 - f. Wählen Sie Lambda-Funktion hinzufügen.

 Note

Sie können andere Lambda-Funktionen aus früheren Modulen entfernen.

Konfigurieren von Abonnements

In diesem Schritt erstellen Sie ein Abonnement, das es dem GG_TrafficLight Shadow ermöglicht, aktualisierte Statusinformationen an die Lambda-Funktion GG_Car_Aggregator zu senden. Das Abonnement wird den Abonnements hinzugefügt, die Sie in [Modul 5](#) erstellt haben, und die alle für dieses Modul erforderlich sind.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Hinzufügen aus.
2. Gehen Sie auf der Seite Abonnement erstellen wie folgt vor:
 - a. Wählen Sie als Quelltyp die Option Service und dann Local Shadow Service aus.
 - b. Wählen Sie als Zieltyp die Option Lambda-Funktion aus, und wählen Sie dann GG_Car_Aggregator.
 - c. Geben Sie unter Topic filter (Themenfilter) **\$aws/things/GG_TrafficLight/shadow/update/documents** ein.
 - d. Wählen Sie Create subscription (Abonnement erstellen) aus.

Dieses Modul erfordert das neue Abonnement sowie die [Abonnements](#), die Sie in Modul 5 erstellt haben.

3. Vergewissern Sie sich, dass der Greengrass-Daemon läuft, wie unter beschrieben. [Bereitstellen von Cloud-Konfigurationen für ein Core-Gerät](#)
4. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Deploy aus.

Testen der Kommunikation

1. Öffnen Sie zwei [Befehlszeilenfenster](#) auf Ihrem Computer. Genau wie in [Modul 5](#) ist ein Fenster für das GG_Switch-Client-Gerät und das andere für das TrafficLight GG_-Client-Gerät. Sie verwenden diese, um die gleichen Befehle wie in Modul 5 auszuführen.

Führen Sie die folgenden Befehle für das GG_Switch-Client-Gerät aus:

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
--cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_Switch
```

Führen Sie die folgenden Befehle für das TrafficLight GG_-Client-Gerät aus:

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName
GG_TrafficLight --clientId GG_TrafficLight
```

Alle 20 Sekunden aktualisiert der Schalter den Schattenzustand mit „G“, „Y“ und „R“ und die Lampe zeigt ihren neuen Zustand an.

2. Der Funktionshandler der Lambda-Funktion wird bei jeder dritten grünen Ampel (alle drei Minuten) ausgelöst, und ein neuer DynamoDB-Datensatz wird erstellt. Gehen `trafficLight.py` Sie nach `lightController.py` drei Minuten Laufzeit zur AWS-Managementkonsole DynamoDB-Konsole und öffnen Sie sie.
3. Wählen Sie im Menü US East (Nord-Virginia) aus AWS-Region . Dies ist die Region, in der die `GG_Car_Aggregator`-Funktion die Tabelle erstellt.
4. Wählen Sie im Navigationsbereich Tabellen und dann die `CarStats`Tabelle aus.
5. Wählen Sie Elemente anzeigen, um die Einträge in der Tabelle anzuzeigen.

Sie sollten die Einträge mit grundlegenden Statistikdaten zu den durchgefahrenen Fahrzeugen sehen (ein Eintrag alle drei Minuten). Möglicherweise müssen Sie die Schaltfläche zum Aktualisieren wählen, um die an der Tabelle vorgenommenen Aktualisierungen anzuzeigen.

6. Wenn der Test nicht erfolgreich ist, überprüfen Sie die Greengrass-Protokolle auf Informationen, die Sie bei der Fehlerbehebung unterstützen können.
 - a. Wechseln Sie zum Root-Benutzer und navigieren Sie zum Verzeichnis `log`. Für den Zugriff auf AWS IoT Greengrass Protokolle sind Root-Rechte erforderlich.

```
sudo su
cd /greengrass/ggc/var/log
```

- b. Prüfen Sie `runtime.log` auf Fehler.

```
cat system/runtime.log | grep 'ERROR'
```

- c. Überprüfen Sie das von der Lambda-Funktion generierte Protokoll.

```
cat user/region/account-id/GG_Car_Aggregator.log
```

Die Skripts `trafficLight.py` und `lightController.py` speichern Verbindungsinformationen im Ordner `groupCA`. Dieser wird im selben Ordner wie die Skripts erstellt. Wenn Sie Verbindungsfehler erhalten, stellen Sie sicher, dass die IP-Adresse in der `ggc-host` Datei mit dem IP-Adressendpunkt für Ihren Core übereinstimmt.

Weitere Informationen finden Sie unter [Fehlerbehebung](#).

Dies ist das Ende des grundlegenden Tutorials. Sie sollten nun das AWS IoT Greengrass Programmiermodell und seine grundlegenden Konzepte verstehen, darunter AWS IoT Greengrass Kerne, Gruppen, Abonnements, Client-Geräte und den Bereitstellungsprozess für Lambda-Funktionen, die am Edge ausgeführt werden.

Sie können die DynamoDB-Tabelle und die Greengrass Lambda-Funktionen und -Abonnements löschen. Um die Kommunikation zwischen dem AWS IoT Greengrass Core-Gerät und der AWS IoT Cloud zu beenden, öffnen Sie ein Terminal auf dem Core-Gerät und führen Sie einen der folgenden Befehle aus:

- Um das AWS IoT Greengrass Kerngerät herunterzufahren:

```
sudo halt
```

- Um den AWS IoT Greengrass Daemon zu stoppen:

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

Modul 7: Simulation der Hardware-Sicherheitsintegration

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Dieses Modul für Fortgeschrittene zeigt Ihnen, wie Sie ein simuliertes Hardware-Sicherheitsmodul (HSM) für die Verwendung mit einem Greengrass Core konfigurieren. Die Konfiguration verwendet SoftHSM. Dabei handelt es sich um eine reine Software-Implementierung mithilfe der [PKCS#11](#) Application Programming Interface (API). Zweck dieses Moduls ist es, Ihnen die Einrichtung einer Umgebung zu ermöglichen, in der Sie lernen und erste Tests in einer reinen Software-Implementierung der PKCS#11-API durchführen können. Das Modul wird nur für Lern- und Ersttests bereitgestellt, nicht für den produktiven Einsatz jeglicher Art.

Mit dieser Konfiguration können Sie mit einem PKCS#11-kompatiblen Service zum Speichern Ihrer privaten Schlüssel experimentieren. Weitere Informationen über die reine Software-Implementierung finden Sie unter [SoftHSM](#). Weitere Informationen zur Integration der Hardwaresicherheit auf einem

AWS IoT Greengrass Core, einschließlich allgemeiner Anforderungen, finden Sie unter [the section called “Integration von Hardware-Sicherheit”](#).

Important

Dieses Modul ist nur für Experimentierzwecke gedacht. Wir raten dringend davon ab, SoftHSM in einer Produktionsumgebung einzusetzen, da dies ein falsches Gefühl von zusätzlicher Sicherheit vermitteln könnte. Die resultierende Konfiguration bietet jedoch keine tatsächlichen Sicherheitsvorteile. Die in SoftHSM gespeicherten Schlüssel sind nicht sicherer als jede andere Art der Geheimhaltung in der Greengrass-Umgebung.

Dieses Modul soll Ihnen ermöglichen, sich über die PKCS#11-Spezifikation zu informieren und erste Tests Ihrer Software durchzuführen, wenn Sie planen, in Zukunft ein echtes hardwarebasiertes HSM einzusetzen.

Sie müssen Ihre zukünftige Hardware-Implementierung separat und vollständig vor jeder Produktionsnutzung testen, da es Unterschiede zwischen der in SoftHSM bereitgestellten PKCS#11-Implementierung und einer hardwarebasierten Implementierung geben kann.

Wenn Sie Hilfe beim Onboarding eines [unterstützten Hardware-Sicherheitsmoduls](#) benötigen, wenden Sie sich an Ihren AWS Enterprise Support-Mitarbeiter.

Bevor Sie beginnen, führen Sie das [Greengrass Device Setup-Skript](#) aus oder stellen Sie sicher, dass Sie [Modul 1](#) und [Modul 2](#) des Lernprogramms „Erste Schritte“ abgeschlossen haben. In diesem Modul gehen wir davon aus, dass Ihr Core bereits bereitgestellt ist und mit ihm kommuniziert. AWS Dieses Modul sollte etwa 30 Minuten in Anspruch nehmen.

Installieren der SoftHSM-Software

In diesem Schritt installieren Sie SoftHSM und die pkcs11-Tools, mit denen Sie Ihre SoftHSM-Instance verwalten.

- Führen Sie in einem Terminal auf Ihrem AWS IoT Greengrass Core-Gerät den folgenden Befehl aus:

```
sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
```

Weitere Informationen zu diesen Paketen finden Sie unter [Installieren von softhsm2](#), [Installieren von libsofthsm2-dev](#) und [Installieren von pkcs11-dump](#).

 Note

Wenn Sie bei der Verwendung dieses Befehls auf Ihrem System auf Probleme stoßen, lesen Sie [SoftHSM Version 2](#) auf GitHub. Diese Seite bietet weitere Installationsinformationen, einschließlich der Anleitung zum Erstellen aus dem Quellcode.

Konfigurieren von SoftHSM

In diesem Schritt [konfigurieren Sie SoftHSM](#).

1. Wechseln Sie zum Root-Benutzer.

```
sudo su
```

2. Verwenden Sie die Handbuchseite, um den systemweiten `softhsm2.conf`-Speicherort zu suchen. Ein häufiger Speicherort ist `/etc/softhsm/softhsm2.conf`, jedoch kann der Speicherort auf einigen Systemen abweichen.

```
man softhsm2.conf
```

3. Erstellen Sie das Verzeichnis für die `softhsm2`-Konfigurationsdatei im systemweiten Speicherort. In diesem Beispiel nehmen wir an, der Speicherort lautet `/etc/softhsm/softhsm2.conf`.

```
mkdir -p /etc/softhsm
```

4. Erstellen Sie das Token-Verzeichnis im Verzeichnis `/greengrass`.

 Note

Wenn dieser Schritt übersprungen wird, meldet `softhsm2` `ERROR: Could not initialize the library`.

```
mkdir -p /greengrass/softhsm2/tokens
```

5. Konfigurieren Sie das Token-Verzeichnis.

```
echo "directories.tokendir = /greengrass/softhsm2/tokens" > /etc/softhsm/softhsm2.conf
```

6. Konfigurieren Sie eine Datei-basierte Backend.

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

Note

Diese Konfigurationseinstellungen sind nur für Experimentierzwecke gedacht. Um alle Konfigurationsoptionen anzuzeigen, lesen Sie die Handbuchseite für die Konfigurationsdatei.

```
man softhsm2.conf
```

Importieren des privaten Schlüssels in SoftHSM

In diesem Schritt initialisieren Sie das SoftHSM-Token, konvertieren das Format des privaten Schlüssels und importieren dann den privaten Schlüssel.

1. Initialisieren Sie das SoftHSM-Token.

```
softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
```

Note

Geben Sie bei Aufforderung den SO-Pin 12345 und den Benutzer-Pin 1234 ein. AWS IoT Greengrass verwendet nicht den SO (Supervisor) Pin, sodass Sie jeden beliebigen Wert verwenden können.

Wenn Sie die Fehlermeldung `CKR_SLOT_ID_INVALID: Slot 0 does not exist` erhalten, versuchen Sie es stattdessen mit dem folgenden Befehl:

```
softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin 1234
```


2. Konvertieren Sie den privaten Schlüssel in ein Format, das vom SoftHSM-Import-Tool verwendet werden kann. Für dieses Tutorial konvertieren Sie den privaten Schlüssel, den Sie von der Option Default Group creation (Standard-Gruppenerstellung) in [Modul 2](#) des Tutorials „Erste Schritte“ erhalten haben.

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -  
out hash.private.pem
```

3. Importieren Sie den privaten Schlüssel in SoftHSM. Führen Sie nur einen der folgenden Befehle aus, je nach Ihrer softhsm2-Version.

Raspbian softhsm2-util v2.2.0-Syntax

```
softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id  
0000 --pin 12340
```

Ubuntu softhsm2-util v2.0.0-Syntax

```
softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin  
1234
```

Dieser Befehl identifiziert den Slot als 0 und definiert das Schlüssel-Label als iotkey. Sie brauchen diese Werte im nächsten Abschnitt.

Nachdem der private Schlüssel importiert wurde, können Sie ihn optional aus dem Verzeichnis /greengrass/certs entfernen. Achten Sie darauf, dass sich die Stammzertifikate für CA und Gerät im Verzeichnis befinden.

Konfigurieren des GreengrassCore für die Verwendung von SoftHSM

In diesem Schritt ändern Sie die Greengrass Core-Konfigurationsdatei für die Verwendung von SoftHSM.

1. Suchen Sie den Pfad zur SoftHSM-Anbieterbibliothek (libsofthsm2.so) auf Ihrem System:
 - a. Holen Sie sich die Liste der installierten Pakete für die Bibliothek.

```
sudo dpkg -L libsofthsm2
```

Die Datei `libsoftthsm2.so` befindet sich im Verzeichnis `softthsm`.

- b. Kopieren Sie den vollständigen Pfad in die Datei (z. B. `/usr/lib/x86_64-linux-gnu/softthsm/libsoftthsm2.so`). Sie benötigen sie später.

2. Stoppen Sie den Greengrass-Daemon.

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

3. Öffnen Sie die Greengrass-Konfigurationsdatei. Dies ist die Datei [config.json](#) im Verzeichnis `/greengrass/config`.

Note

Die Beispiele in diesem Verfahren sind unter der Annahme verfasst, dass die Datei `config.json` das Format verwendet, das aus der Option Easy Group creation (Einfache Gruppenerstellung) in [Modul 2](#) des Tutorials „Erste Schritte“ erzeugt wird.

4. Fügen Sie in das Objekt `crypto.principals` das folgende MQTT Server-Zertifikat-Objekt ein. Fügen Sie bei Bedarf ein Komma hinzu, um eine gültige JSON-Datei zu erstellen.

```
"MQTTServerCertificate": {  
  "privateKeyPath": "path-to-private-key"  
}
```

5. Fügen Sie in das Objekt `crypto` das folgende PKCS11-Objekt ein. Fügen Sie bei Bedarf ein Komma hinzu, um eine gültige JSON-Datei zu erstellen.

```
"PKCS11": {  
  "P11Provider": "/path-to-pkcs11-provider-so",  
  "slotLabel": "crypto-token-name",  
  "slotUserPin": "crypto-token-user-pin"  
}
```

Die Datei sollte wie folgt aussehen:

```
{  
  "coreThing" : {  
    "caPath" : "root.ca.pem",  
    "certPath" : "hash.cert.pem",
```

```

    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}

```

Note

Um over-the-air (OTA-) Updates mit Hardwaresicherheit verwenden zu können, muss das PKCS11 Objekt auch die OpenSSLEngine Eigenschaft enthalten. Weitere Informationen finden Sie unter [the section called “Konfigurieren von OTA-Updates”](#).

6. Bearbeiten Sie das Objekt crypto:

- a. Konfigurieren Sie das Objekt PKCS11.
 - Geben Sie für P11Provider den vollständigen Pfad zur Datei libsoftsm2.so ein.
 - Geben Sie unter slotLabel den Wert greengrass ein.
 - Geben Sie unter slotUserPin den Wert 1234 ein.
- b. Konfigurieren Sie die privaten Schlüsselpfade im Objekt principals. Bearbeiten Sie die Eigenschaft certificatePath nicht.
 - Geben Sie für die Eigenschaften privateKeyPath den folgenden RFC 7512 PKCS#11-Pfad ein (der die Bezeichnung des Schlüssels angibt). Machen Sie dies für die Prinzipale von IoTCertificate, SecretsManager und MQTTServerCertificate.

```
pkcs11:object=iotkey;type=private
```

- c. Prüfen Sie das crypto-Objekt. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
"crypto": {
  "PKCS11": {
    "P11Provider": "/usr/lib/x86_64-linux-gnu/softsm/libsoftsm2.so",
    "slotLabel": "greengrass",
    "slotUserPin": "1234"
  },
  "principals": {
    "MQTTServerCertificate": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "SecretsManager": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "IoTCertificate": {
      "certificatePath": "file://certs/core.crt",
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    }
  },
  "caPath": "file://certs/root.ca.pem"
}
```

7. Entfernen Sie die Werte für caPath, certPath und keyPath aus dem coreThing-Objekt. Das sollte bei Ihnen ähnlich wie im folgenden Bild aussehen:

```
"coreThing" : {  
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",  
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",  
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",  
  "keepAlive" : 600  
}
```

Note

Für dieses Tutorial geben Sie für alle Prinzipale den gleichen privaten Schlüssel an. Weitere Informationen zur Auswahl des privaten Schlüssels für den lokalen MQTT-Server finden Sie unter [Performance](#). Weitere Informationen über den Manager lokaler Secrets finden Sie unter [Bereitstellen von Secrets für den Core](#).

Testen der Konfiguration

- Halten Sie den Greengrass-Daemon an.

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

Wenn der Daemon erfolgreich startet, dann ist Ihr Core korrekt konfiguriert.

Sie sind nun bereit, die PKCS#11-Spezifikation kennenzulernen und erste Tests mit der PKCS#11-API durchzuführen, die von der SoftHSM-Implementierung bereitgestellt wird.

Important

Auch hier ist es äußerst wichtig zu wissen, dass dieses Modul nur zum Lernen und Testen gedacht ist. Es erhöht nicht wirklich den Sicherheitsstatus Ihrer Greengrass-Umgebung.

Stattdessen soll das Modul Ihnen ermöglichen, mit dem Lernen und Testen zu beginnen, um sich auf den Einsatz eines echten hardwarebasierten HSM in der Zukunft vorzubereiten. Zu diesem Zeitpunkt müssen Sie Ihre Software vor jeder Produktionsnutzung separat und vollständig mit dem hardwarebasierten HSM testen, da

es Unterschiede zwischen der in SoftHSM bereitgestellten PKCS#11-Implementierung und einer hardwarebasierten Implementierung geben kann.

Weitere Informationen finden Sie auch unter

- PKCS #11 Kryptographische Token-Schnittstelle, Bedienungsanleitung Version 2.40.
Herausgegeben von John Leiseboer und Robert Griffin. 16. November 2014. Anmerkung 02 des OASIS-Ausschusses. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html>. Letzte Version: <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- [RFC 7512](#)

OTA-Updates der AWS IoT Greengrass Core-Software

Das AWS IoT Greengrass Core-Softwarepaket enthält einen Update-Agenten, der AWS IoT Greengrass Softwareupdates over-the-air (OTA) durchführen kann. Sie können OTA-Updates verwenden, um die neueste Version der AWS IoT Greengrass Core-Software oder der OTA-Update-Agent-Software auf einem oder mehreren Kernen zu installieren. Mit OTA-Updates müssen Ihre Core-Geräte nicht physisch vorhanden sein.

Wir empfehlen, wenn möglich OTA-Updates zu verwenden. Sie stellen einen Mechanismus bereit, mit dem Sie den Aktualisierungsstatus und den Aktualisierungsverlauf nachverfolgen können. Wenn ein Update fehlschlägt, setzt der OTA-Update-Agent auf die vorherige Softwareversion zurück.

Note

OTA-Updates werden nicht unterstützt, wenn Sie `apt` zur Installation der AWS IoT Greengrass Core-Software verwenden. Für diese Installationen wird empfohlen, die Software mithilfe von `apt` zu aktualisieren. Weitere Informationen finden Sie unter [the section called “Installieren aus einem APT-Repository”](#).

OTA-Updates machen Folgendes effizienter:

- Schließen von Sicherheitslücken.
- Behebung von Software-Stabilitätsproblemen.
- Bereitstellung neuer oder verbesserter Funktionen.

Diese Funktion lässt sich in [AWS IoT -Aufgaben](#) integrieren.

Voraussetzungen

Die folgenden Anforderungen gelten für AWS IoT Greengrass OTA-Softwareupdates.

- Für den Greengrass-Kern müssen mindestens 400 MB Festplattenspeicher im lokalen Speicher verfügbar sein. Der OTA-Update-Agent benötigt etwa das Dreifache der Runtime-Nutzungsanforderungen der AWS IoT Greengrass Core-Software. Weitere Informationen finden Sie unter [Service quotas \(Servicekontingente\)](#) für den Greengrass-Core im Allgemeine Amazon Web Services-Referenz.

- Der Greengrass-Kern muss eine Verbindung zum AWS Cloud haben.
- Der Greengrass-Core muss korrekt konfiguriert und mit Zertifikaten und Schlüsseln für die Authentifizierung mit AWS IoT Core und ausgestattet sein. AWS IoT Greengrass Weitere Informationen finden Sie unter [the section called “X.509-Zertifikate”](#).
- Der Greengrass-Core kann nicht so konfiguriert werden, dass er einen Netzwerk-Proxy verwendet.

 Note

Ab AWS IoT Greengrass Version 1.9.3 werden OTA-Updates auf Kernen unterstützt, die den MQTT-Verkehr so konfigurieren, dass er Port 443 anstelle des Standardports 8883 verwendet. Der OTA-Update-Agent unterstützt jedoch keine Updates über einen Netzwerk-Proxy. Weitere Informationen finden Sie unter [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).

- Trusted Boot kann auf der Partition, die die AWS IoT Greengrass Core-Software enthält, nicht aktiviert werden.

 Note

Sie können die AWS IoT Greengrass Core-Software auf einer Partition installieren und ausführen, für die Trusted Boot aktiviert ist, OTA-Updates werden jedoch nicht unterstützt.

- AWS IoT Greengrass muss über read/write Berechtigungen für die Partition verfügen, die die AWS IoT Greengrass Core-Software enthält.
- Wenn Sie ein Init-System zur Verwaltung Ihres Greengrass-Cores verwenden, müssen Sie OTA-Updates zur Integration in das Init-System konfigurieren. Weitere Informationen finden Sie unter [the section called “Integration in Init-Systeme”](#).
- Sie müssen eine Rolle erstellen, die verwendet wird, um Amazon S3 URLs für AWS IoT Greengrass Softwareupdate-Artefakte vorzukonfigurieren. Diese Unterzeichnerrolle ermöglicht AWS IoT Core den Zugriff auf Software-Update-Artefakte, die in Amazon S3 in Ihrem Namen gespeichert sind. Weitere Informationen finden Sie unter [the section called “IAM-Berechtigungen für OTA-Updates”](#).

IAM-Berechtigungen für OTA-Updates

Wenn eine neue Version der AWS IoT Greengrass Core-Software AWS IoT Greengrass veröffentlicht wird, werden die in Amazon S3 gespeicherten Softwareartefakte AWS IoT Greengrass aktualisiert, die für das OTA-Update verwendet werden.

Sie AWS-Konto müssen eine Amazon S3 S3-URL-Unterzeichnerrolle angeben, die für den Zugriff auf diese Artefakte verwendet werden kann. Die Rolle muss über eine Berechtigungsrichtlinie verfügen, die die `s3:GetObject` Aktion für die Buckets in Ziel s AWS-Region ermöglicht. Die Rolle muss auch über eine Vertrauensrichtlinie verfügen, mit der `iot.amazonaws.com` die Rolle als vertrauenswürdige Entität übernehmen kann.

Berechtigungsrichtlinie

Für Rollenberechtigungen können Sie die AWS verwaltete Richtlinie verwenden oder eine benutzerdefinierte Richtlinie erstellen.

- Verwenden Sie die AWS verwaltete Richtlinie

Die von [Greengrass OTAUpdate ArtifactAccess](#) verwaltete Richtlinie wird bereitgestellt von AWS IoT Greengrass. Verwenden Sie diese Richtlinie, wenn Sie den Zugriff in allen aktuell und in future unterstützten AWS IoT Greengrass Amazon Web Services Services-Regionen zulassen möchten.

- Erstellen Sie eine benutzerdefinierte Richtlinie

Sie sollten eine benutzerdefinierte Richtlinie erstellen, wenn Sie die Amazon Web Services Services-Regionen, in denen Ihre Cores bereitgestellt werden, explizit angeben möchten. Die folgende Beispielrichtlinie ermöglicht den Zugriff auf AWS IoT Greengrass Softwareupdates in sechs Regionen.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ]
    }
  ]
}
```

```

    "Resource": [
      "arn:aws:s3:::us-east-1-greengrass-updates/*",
      "arn:aws:s3:::us-west-2-greengrass-updates/*",
      "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
      "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
      "arn:aws:s3:::eu-central-1-greengrass-updates/*",
      "arn:aws:s3:::eu-west-1-greengrass-updates/*"
    ]
  }
}

```

Vertrauensrichtlinie

Die der Rolle angefügte Vertrauensrichtlinie muss die `sts:AssumeRole`-Aktion zulassen und `iot.amazonaws.com` als Prinzipal definieren. Dies ermöglicht es AWS IoT Core, die Rolle als vertrauenswürdige Entität zu übernehmen. Hier ist ein Beispielrichtliniendokument:

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowIotToAssumeRole",
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Effect": "Allow"
    }
  ]
}

```

Darüber hinaus muss der Benutzer, der ein OTA-Update initiiert, über Berechtigungen zum Verwenden von `greengrass:CreateSoftwareUpdateJob` und `iot:CreateJob` verfügen und zum Übergeben der Berechtigungen der Signer-Rolle `iam:PassRole` verwenden. Hier ist ein Beispiel für eine IAM-Richtlinie:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "greengrass:CreateSoftwareUpdateJob"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateJob"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/name-of-s3-url-signer-
role"
    }
  ]
}
```

Überlegungen

Berücksichtigen Sie vor einer OTA-Aktualisierung der Greengrass Core-Software die Auswirkungen auf die Geräte in Ihrer Greengrass-Gruppe, sowohl das Core-Gerät als auch die Client-Geräte, die lokal mit dem Core verbunden sind:

- Der Core wird während der Aktualisierung heruntergefahren.
- Alle auf dem Core ausgeführten Lambda-Funktionen werden beendet. Wenn diese Funktionen Daten auf lokale Ressourcen schreiben, kann es sein, dass diese Ressourcen in einem fehlerhaften Status verbleiben, wenn sie nicht ordnungsgemäß heruntergefahren werden.

- Während der Ausfallzeit des Kerns gehen alle Verbindungen zum AWS Cloud verloren. Von Client-Geräten über den Core geleitete Nachrichten gehen verloren.
- Zwischenspeicherungen von Anmeldeinformationen gehen verloren.
- Warteschlangen, in denen anstehende Aufgaben für Lambda-Funktionen enthalten sind, gehen verloren.
- Langlebige Lambda-Funktionen verlieren ihre dynamischen Statusinformationen und alle ausstehenden Aufgaben werden gelöscht.

Die folgenden Statusinformationen bleiben während einer OTA-Aktualisierung erhalten:

- Core-Konfiguration
- Konfiguration der Greengrass-Gruppe
- Lokale Schatten
- Greengrass-Protokolle
- Protokolle des OTA-Update-Agents

Greengrass OTA-Update-Agent

Der Greengrass OTA Update Agent ist die Softwarekomponente auf dem Gerät, die in der Cloud erstellte und bereitgestellte Update-Jobs verarbeitet. Der OTA-Update-Agent wird im selben Softwarepaket wie die AWS IoT Greengrass Core-Software vertrieben. Der Agent befindet sich in `/greengrass-root/ota/ota_agent/ggc-ota`. Er schreibt Protokolle in `/var/log/greengrass/ota/ggc_ota.txt`.

Note

`greengrass-root` stellt den Pfad dar, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

Sie können den OTA-Update-Agenten starten, indem Sie die Binärdatei manuell ausführen oder sie als Teil eines Init-Skripts, z. B. einer Systemd-Servicedatei, integrieren. Wenn Sie die Binärdatei manuell ausführen, sollte sie als Root ausgeführt werden. Beim Start wartet der OTA-Update-Agent auf AWS IoT Greengrass Softwareupdate-Jobs von AWS IoT Core und führt sie nacheinander aus. Der OTA-Update-Agent ignoriert alle anderen Jobtypen. AWS IoT

Der folgende Auszug zeigt ein Beispiel für eine Systemd-Dienstdatei zum Starten, Stoppen und Neustarten des OTA-Update-Agents:

```
[Unit]
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota

[Install]
WantedBy=multi-user.target
```

Ein Core, der das Ziel eines Updates ist, darf nicht zwei Instanzen des OTA-Update-Agents ausführen. Andernfalls verarbeiten die beiden Agenten dieselben Aufträge, was zu Konflikten führt.

Integration in Init-Systeme

Während eines OTA-Updates startet der OTA-Update-Agent die Binärdateien auf dem Kerngerät neu. Wenn die Binärdateien ausgeführt werden, kann dies zu Konflikten führen, wenn ein Init-System den Status der AWS IoT Greengrass Core-Software oder des Agenten während des Updates überwacht. Um die Integration des OTA-Aktualisierungsmechanismus mit Ihren Init-Überwachungsstrategien zu unterstützen, können Sie Shell-Skripts schreiben, die vor und nach einer Aktualisierung ausgeführt werden. Sie können das `ggc_pre_update.sh` Skript beispielsweise verwenden, um Daten zu sichern oder Prozesse zu beenden, bevor das Gerät heruntergefahren wird.

Um den OTA-Update-Agenten anzuweisen, diese Skripts auszuführen, müssen Sie das `"managedRespawn" : true` Flag in der [Datei `config.json` angeben](#). Diese Einstellung wird im folgenden Auszug dargestellt:

```
{
  "coreThing": {
    ...
  },
  "runtime": {
    ...
  },
  "managedRespawn": true
  ...
}
```

```
}
```

Managed Respawn mit OTA-Updates

Die folgenden Anforderungen gelten für OTA-Updates mit der `managedRespawn` Einstellung auf: `true`

- Die folgenden Shell-Skripte müssen im `/greengrass-root/usr/scripts` Verzeichnis vorhanden sein:
 - `ggc_pre_update.sh`
 - `ggc_post_update.sh`
 - `ota_pre_update.sh`
 - `ota_post_update.sh`
- Die Skripte müssen einen erfolgreichen Rückgabecode zurückgeben.
- Die Skripte müssen sich im Besitz von `root` befinden und nur von `root` ausführbar sein.
- Das `ggc_pre_update.sh` Skript muss den Greengrass-Daemon stoppen.
- Das `ggc_post_update.sh` Skript muss den Greengrass-Daemon starten.

Note

Da der OTA-Aktualisierungsagent seinen eigenen Prozess verwaltet, müssen die `ota_post_update.sh` Skripte `ota_pre_update.sh` und den OTA-Dienst nicht beenden oder starten.

Der OTA-Update-Agent führt die Skripts vom aus `/greengrass-root/usr/scripts`. Die Verzeichnisstruktur sollte wie folgt aussehen:

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
```

```
| |-- ota_pre_update.sh  
| |-- ota_post_update.sh  
|-- ota
```

Wenn auf gesetzt `managedRespawn` ist `true`, überprüft der OTA-Update-Agent das `/greengrass-root/usr/scripts` Verzeichnis vor und nach dem Softwareupdate auf diese Skripts. Wenn die Skripts nicht existieren, schlägt das Update fehl. AWS IoT Greengrass validiert den Inhalt dieser Skripts nicht. Es hat sich bewährt, sicherzustellen, dass Ihre Skripts korrekt funktionieren, und geben Sie bei Fehlern entsprechende Exit-Codes aus.

Für OTA-Updates der AWS IoT Greengrass Core-Software:

- Vor dem Start des Updates führt der Agent das `ggc_pre_update.sh`-Skript aus. Verwenden Sie dieses Skript für Befehle, die ausgeführt werden müssen, bevor der OTA-Update-Agent das AWS IoT Greengrass Core-Softwareupdate startet, z. B. um Daten zu sichern oder laufende Prozesse zu beenden. Das folgende Beispiel zeigt ein einfaches Skript zum Stoppen des Greengrass-Daemons.

```
#!/bin/bash  
set -euo pipefail  
systemctl stop greengrass
```

- Nach Abschluss des Updates führt der Agent das `ggc_post_update.sh`-Skript aus. Verwenden Sie dieses Skript für Befehle, die ausgeführt werden müssen, nachdem der OTA-Update-Agent das AWS IoT Greengrass Core-Softwareupdate gestartet hat, z. B. um Prozesse neu zu starten. Das folgende Beispiel zeigt ein einfaches Skript zum Starten des Greengrass-Daemons.

```
#!/bin/bash  
set -euo pipefail  
systemctl start greengrass
```

Für OTA-Updates des OTA-Update-Agents:

- Vor dem Start des Updates führt der Agent das `ota_pre_update.sh`-Skript aus. Verwenden Sie dieses Skript für Befehle, die ausgeführt werden müssen, bevor sich der OTA-Update-Agent selbst aktualisiert, z. B. um Daten zu sichern oder laufende Prozesse zu beenden.
- Nach Abschluss des Updates führt der Agent das `ota_post_update.sh`-Skript aus. Verwenden Sie dieses Skript für Befehle, die ausgeführt werden müssen, nachdem sich der OTA-Update-Agent selbst aktualisiert hat, z. B. um Prozesse neu zu starten.

 Note

Wenn auf gesetzt `managedRespawn` ist `false`, führt der OTA-Update-Agent die Skripts nicht aus.

Erstellen eines OTA-Updates

Gehen Sie wie folgt vor, um ein OTA-Update der AWS IoT Greengrass Software auf einem oder mehreren Kernen durchzuführen:

1. Stellen Sie sicher, dass Ihre Cores die [Anforderungen](#) für OTA-Updates erfüllen.

 Note

Wenn Sie ein Init-System für die Verwaltung der AWS IoT Greengrass Core-Software oder des OTA-Update-Agents konfiguriert haben, überprüfen Sie Folgendes auf Ihren Kernen:

- Die [config.json](#)-Datei legt "managedRespawn" : `true` fest.
- Das `/greengrass-root/usr/scripts`-Verzeichnis enthält folgende Skripte:
 - `ggc_pre_update.sh`
 - `ggc_post_update.sh`
 - `ota_pre_update.sh`
 - `ota_post_update.sh`

Weitere Informationen finden Sie unter [the section called "Integration in Init-Systeme"](#).

2. Starten Sie in einem Core-Geräteterminal den OTA-Update-Agent.

```
cd /greengrass-root/ota/ota_agent
sudo ./ggc-ota
```


 Note

greengrass-root steht für den Pfad, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis /greengrass.

Starten Sie nicht mehrere Instanzen des OTA-Update-Agents auf einem Core, da dies zu Konflikten führen kann.

3. Verwenden Sie die AWS IoT Greengrass API, um einen Softwareupdate-Job zu erstellen.
 - a. Rufen Sie die [CreateSoftwareUpdateJob](#)-API auf. In diesem Beispielverfahren verwenden wir AWS CLI Befehle.

Der folgende Befehl erstellt einen Job, der die AWS IoT Greengrass Core-Software auf einem Kern aktualisiert. Ersetzen Sie die Beispielwerte und führen Sie dann den Befehl aus.

Linux or macOS terminal

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

Windows command prompt

```
aws greengrass create-software-update-job ^
--update-targets-architecture x86_64 ^
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] ^
--update-targets-operating-system ubuntu ^
--software-to-update core ^
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^
--update-agent-log-level WARN ^
--amzn-client-token myClientToken1
```

Dieser Befehl gibt die folgende Antwort zurück.

```
{
  "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "IotJobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "PlatformSoftwareVersion": "1.10.1"
}
```

- b. Kopieren Sie `IotJobId` aus der Antwort.
- c. Rufen Sie [DescribeJob](#) die AWS IoT Core API auf, um den Jobstatus zu sehen. Ersetzen Sie den Beispielpunkt durch Ihre Aufgaben-ID, und führen Sie dann den Befehl aus.

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-
a1da0EXAMPLE
```

Der Befehl gibt ein Antwortobjekt zurück, das Informationen über die Aufgabe enthält, einschließlich `status` und `jobProcessDetails`.

```
{
  "job": {
    "jobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "targetSelection": "SNAPSHOT",
    "status": "IN_PROGRESS",
    "targets": [
      "arn:aws:iot:region:123456789012:thing/myCoreDevice"
    ],
    "description": "This job was created by Greengrass to update the
Greengrass Cores in the targets with version 1.10.1 of the core software
running on x86_64 architecture.",
    "presignedUrlConfig": {
      "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
      "expiresInSec": 3600
    },
    "jobExecutionsRolloutConfig": {},
    "createdAt": 1588718249.079,
    "lastUpdatedAt": 1588718253.419,
    "jobProcessDetails": {
```

```
        "numberOfCanceledThings": 0,  
        "numberOfSucceededThings": 0,  
        "numberOfFailedThings": 0,  
        "numberOfRejectedThings": 0,  
        "numberOfQueuedThings": 1,  
        "numberOfInProgressThings": 0,  
        "numberOfRemovedThings": 0,  
        "numberOfTimedOutThings": 0  
    },  
    "timeoutConfig": {}  
}  
}
```

Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

CreateSoftwareUpdateJob API

Sie können die `CreateSoftwareUpdateJob` API verwenden, um die AWS IoT Greengrass Core-Software oder die OTA-Update-Agent-Software auf Ihren Core-Geräten zu aktualisieren. Diese API erstellt eine AWS IoT -Snapshot-Aufgabe, die Geräte benachrichtigt, wenn ein Update verfügbar ist. Nach dem Anruf können Sie andere AWS IoT Jobbefehle verwenden `CreateSoftwareUpdateJob`, um das Softwareupdate zu verfolgen. Weitere Informationen finden Sie unter [Jobs](#) im AWS IoT Developer Guide.

Das folgende Beispiel zeigt, wie Sie mit AWS CLI dem einen Job erstellen, der die AWS IoT Greengrass Core-Software auf einem Core-Gerät aktualisiert:

```
aws greengrass create-software-update-job \  
--update-targets-architecture x86_64 \  
--update-targets ["arn:aws:iot:region:123456789012:thing/myCoreDevice\""] \  
--update-targets-operating-system ubuntu \  
--software-to-update core \  
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \  
--update-agent-log-level WARN \  
--amzn-client-token myClientToken1
```

Der Befehl `create-software-update-job` gibt eine JSON-Antwort zurück, die die Auftrags-ID, den Auftrags-ARN und die Softwareversion enthält, die durch das Update installiert wurden:

```
{
  "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
  "PlatformSoftwareVersion": "1.9.2"
}
```

Schritte, die Ihnen zeigen, wie `create-software-update-job` zum Aktualisieren eines Core-Geräts verwendet wird, finden Sie unter [the section called “Erstellen eines OTA-Updates”](#).

Der Befehl `create-software-update-job` hat die folgenden Parameter:

`--update-targets-architecture`

Die Architektur des Core-Geräts.

Gültige Werte: `armv7l`, `armv6l`, `x86_64` oder `aarch64`

`--update-targets`

Cores, die aktualisiert werden sollen. Die Liste kann einzelne Kerne und ARNs Dinggruppen enthalten ARNs , deren Mitglieder Kerne sind. Weitere Informationen zu Dinggruppen finden Sie unter [Statische Dinggruppen](#) im AWS IoT Entwicklerhandbuch.

`--update-targets-operating-system`

Das Betriebssystem des Core-Geräts.

Gültige Werte: `ubuntu`, `amazon_linux`, `raspbian` oder `openwrt`

`--software-to-update`

Gibt an, ob die Software des Kerns oder die OTA-Update-Agent-Software aktualisiert werden soll.

Gültige Werte: `core` oder `ota_agent`.

`--s3-url-signer-role`

Der ARN der IAM-Rolle, mit der die Amazon S3 S3-URL, die auf die AWS IoT Greengrass Softwareupdate-Artefakte verweist, vorab signiert wurde. Die der Rolle beigefügte Berechtigungsrichtlinie muss die `s3:GetObject` Aktion für die Buckets in den Ziel-s zulassen. AWS-Region Die Rolle muss auch `iot.amazonaws.com` erlauben, die Rolle als vertrauenswürdige Entität zu übernehmen. Weitere Informationen finden Sie unter [the section called “IAM-Berechtigungen für OTA-Updates”](#).

--amzn-client-token

(Optional) Ein Client-Token für idempotente Anfragen. Geben Sie einen eindeutigen Token an, um zu verhindern, dass aufgrund interner wiederholter Versuche doppelte Aktualisierungen durchgeführt werden.

--update-agent-log-level

(Optional) Die Protokollierungsebene für Protokollanweisungen, die vom OTA-Update-Agent generiert wurden. Der Standardwert ist ERROR.

Gültige Werte: NONE, TRACE, DEBUG, VERBOSE, INFO, WARN, ERROR oder FATAL

 Note

CreateSoftwareUpdateJob akzeptiert nur Anforderungen für die folgenden unterstützten Architektur- und Betriebssystemkombinationen:

- ubuntu/x86_64
- ubuntu/aarch64
- amazon_linux/x86_64
- raspbian/armv7l
- raspbian/armv6l
- openwrt/aarch64
- openwrt/armv7l

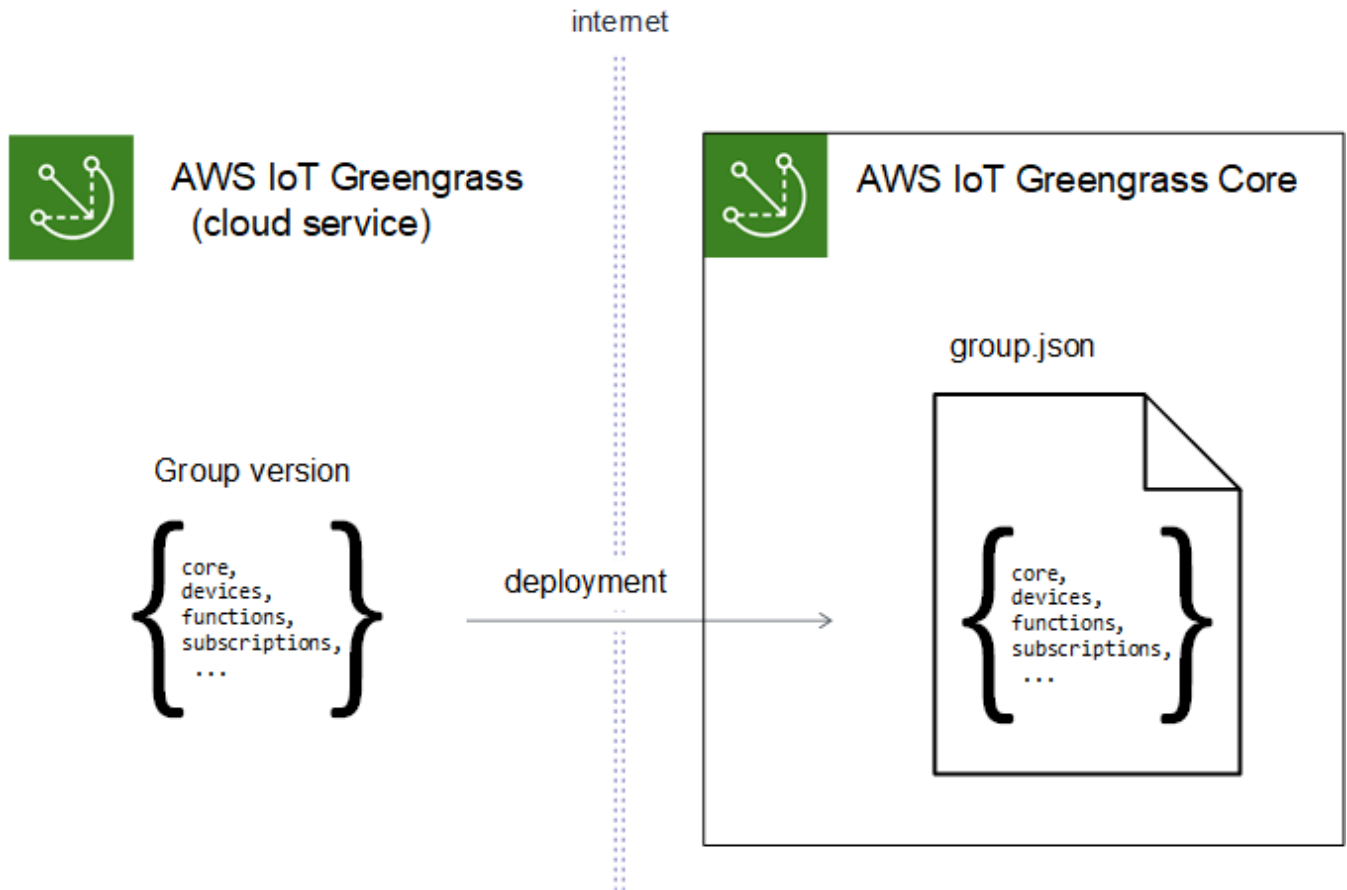
AWS IoT Greengrass Gruppen für einen AWS IoT Greengrass Kern bereitstellen

Verwenden Sie AWS IoT Greengrass Gruppen, um Entitäten in Ihrer Edge-Umgebung zu organisieren. Sie verwenden Gruppen auch, um zu steuern, wie die Entitäten in der Gruppe miteinander und mit den interagieren AWS Cloud. Beispielsweise werden nur die Lambda-Funktionen in der Gruppe für die lokale Ausführung bereitgestellt, und nur die Geräte in der Gruppe können über den lokalen MQTT-Server kommunizieren.

Eine Gruppe muss einen [Core](#) enthalten, bei dem es sich um ein AWS IoT Gerät handelt, auf dem die AWS IoT Greengrass Core-Software ausgeführt wird. Der Core fungiert als Edge-Gateway und bietet AWS IoT Core Funktionen in der Edge-Umgebung. Abhängig von Ihren geschäftlichen Anforderungen können Sie auch die folgenden Entitäten zu einer Gruppe hinzufügen:

- Client-Geräte. Dargestellt als Objekte in der AWS IoT -Registrierung. Diese Geräte müssen [FreeRTOS](#) ausführen oder das [AWS IoT Device SDK](#) oder die [AWS IoT Greengrass Discovery-API](#) verwenden, um Verbindungsinformationen für den Core abzurufen. Nur Client-Geräte, die Mitglieder der Gruppe sind, können eine Verbindung zum Core herstellen.
- Lambda-Funktionen. Benutzerdefinierte serverlose Anwendungen, die Code auf dem Kern ausführen. Lambda-Funktionen werden in einer Greengrass-Gruppe verfasst AWS Lambda und von dort aus referenziert. Weitere Informationen finden Sie unter [Lokale Lambda-Funktionen ausführen](#).
- Konnektoren Vordefinierte serverlose Anwendungen, die Code auf dem Kern ausführen. Konnektoren können eine integrierte Integration mit der lokalen Infrastruktur, Geräteprotokollen und anderen Cloud-Diensten ermöglichen. AWS Weitere Informationen finden Sie unter [Integrieren von Services und Protokollen mit Konnektoren](#).
- Abonnements Definiert die Herausgeber, Abonnenten und MQTT-Themen (oder Themen), die für die MQTT-Kommunikation autorisiert sind.
- Ressourcen Verweise auf lokale [Geräte und Volumes](#), [Modelle für maschinelles Lernen](#) und [Geheimnisse](#), die von den Funktionen und Konnektoren von Greengrass Lambda für die Zugriffskontrolle verwendet werden.
- Protokolle. Protokollierung von Konfigurationen für AWS IoT Greengrass Systemkomponenten und Lambda-Funktionen. Weitere Informationen finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

Sie verwalten Ihre Greengrass-Gruppe im AWS Cloud und stellen sie dann in einem Core bereit. Die Bereitstellung kopiert die Gruppenkonfiguration in die `group.json`-Datei auf dem Core-Gerät. Diese Datei befindet sich unter ***greengrass-root***/`ggc/deployments/group`.



Note

Während einer Bereitstellung stoppt der Greengrass-Daemon-Prozess auf dem Core-Gerät und startet dann neu.

Gruppen von der Konsole aus bereitstellen AWS IoT

Sie können eine Gruppe über die Konfigurationsseite der Gruppe in der AWS IoT Konsole bereitstellen und ihre Bereitstellungen verwalten.

 Note

Um diese Seite in der Konsole zu öffnen, wählen Sie Greengrass-Geräte, dann Gruppen (V1) und dann unter Greengrass-Gruppen Ihre Gruppe aus.

Bereitstellen der aktuellen Version der Gruppe

- Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Bereitstellen aus.

Anzeigen des Bereitstellungsverlaufs der Gruppe

Der Bereitstellungsverlauf einer Gruppe enthält das Datum und die Uhrzeit, die Gruppenversion und den Status der einzelnen Bereitstellungsversuche.

1. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Registerkarte Bereitstellungen aus.
2. Um weitere Informationen zu einer Bereitstellung, einschließlich Fehlermeldungen, zu sehen, wählen Sie in der AWS IoT Konsole unter Greengrass-Geräte die Option Deployments aus.

Erneutes Bereitstellen einer Gruppenbereitstellung

Sie können eine Bereitstellung erneut bereitstellen, wenn die aktuelle Bereitstellung fehlschlägt oder auf eine andere Gruppenversion zurückgesetzt wird.

1. Wählen Sie in der AWS IoT Konsole Greengrass-Geräte und dann Gruppen (V1) aus.
2. Wählen Sie die Registerkarte Bereitstellen.
3. Wählen Sie die Bereitstellung aus, die Sie erneut bereitstellen möchten, und wählen Sie Erneut bereitstellen.

Zurücksetzen von Gruppenbereitstellungen

Sie können Gruppenbereitstellungen zurücksetzen, um eine Gruppe zu verschieben oder zu löschen oder Bereitstellungsinformationen zu entfernen. Weitere Informationen finden Sie unter [the section called “Zurücksetzen von Bereitstellungen”](#).

1. Wählen Sie in der AWS IoT Konsole Greengrass-Geräte und dann Gruppen (V1) aus.
2. Wählen Sie die Registerkarte Bereitstellen.

3. Wählen Sie die Bereitstellung aus, die Sie zurücksetzen möchten, und wählen Sie Bereitstellungen zurücksetzen.

Bereitstellen von Gruppen mit der API AWS IoT Greengrass

Die AWS IoT Greengrass API bietet die folgenden Aktionen zum Bereitstellen von AWS IoT Greengrass Gruppen und zum Verwalten von Gruppenbereitstellungen. Sie können diese Aktionen über die AWS CLI AWS IoT Greengrass API oder das AWS SDK aufrufen.

Aktion	Beschreibung
CreateDeployment	Erstellt eine NewDeployment - oder Redeployment -Bereitstellung. Möglicherweise möchten Sie eine Bereitstellung erneut bereitstellen, wenn die aktuelle Bereitstellung fehlschlägt. Sie können auch eine erneute Bereitstellung durchführen, um eine andere Gruppenversion wiederherzustellen.
GetDeploymentStatus	Gibt den Status einer Bereitstellung zurück: Building, InProgress , Success oder Failure. Sie können Amazon EventBridge Events so konfigurieren, dass Sie Bereitstellungsbenachrichtigungen erhalten. Weitere Informationen finden Sie unter the section called “Abrufen von Bereitstellungsbenachrichtigungen”.
ListDeployments	Gibt den Bereitstellungsverlauf für die Gruppe zurück.
ResetDeployments	Setzt die Bereitstellungen für die Gruppe zurück.

Aktion	Beschreibung
	Sie können Gruppenbereitstellungen zurücksetzen, um eine Gruppe zu verschieben oder zu löschen oder Bereitstellungsinformationen zu entfernen. Weitere Informationen finden Sie unter the section called “Zurücksetzen von Bereitstellungen” .

Note

Weitere Informationen zu Massenbereitstellungsvorgängen finden Sie unter [the section called “Erstellen von Sammelbereitstellungen”](#).

Abrufen der Gruppen-ID

Die Gruppen-ID wird häufig in API-Aktionen verwendet. Sie können die [ListGroupsWithName](#)-Aktion verwenden, um die ID der Zielgruppe aus Ihrer Gruppenliste zu finden. Verwenden Sie AWS CLI beispielsweise in der `list-groups` Befehl.

```
aws greengrass list-groups
```

Sie können auch die `query`-Option zum Filtern der Ergebnisse einschließen. Zum Beispiel:

- So rufen Sie die zuletzt erstellte Gruppe ab:

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))[0]"
```

- So rufe Sie eine Gruppe anhand des Namens ab:

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

Nachfolgend finden Sie eine `list-groups`-Beispielantwort. Die Informationen für jede Gruppe beinhalten die ID der Gruppe (in der `Id`-Eigenschaft) und die ID der letzten Gruppenversion (in der `LatestVersion`-Eigenschaft). Um eine andere Version IDs für eine Gruppe zu erhalten, verwenden Sie die Gruppen-ID mit [ListGroupVersions](#).

 Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
  ]
}
```

Wenn Sie keine angeben AWS-Region, verwenden AWS CLI Befehle die Standardregion aus Ihrem Profil. Um Gruppen in einer anderen Region zurückzugeben, fügen Sie die *region* Option hinzu. Zum Beispiel:

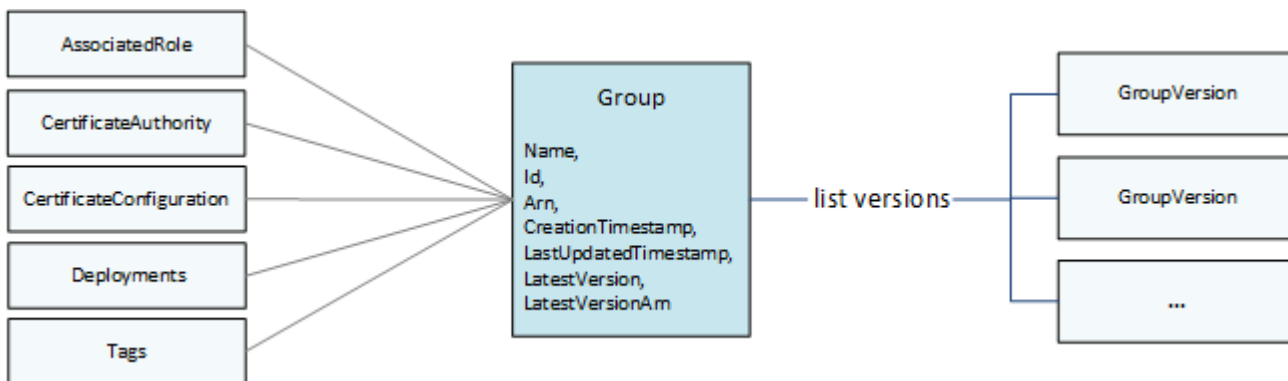
```
aws greengrass list-groups --region us-east-1
```

Überblick über das AWS IoT Greengrass Gruppenobjektmodell

Bei der Programmierung mit der AWS IoT Greengrass API ist es hilfreich, das Greengrass-Gruppenobjektmodell zu verstehen.

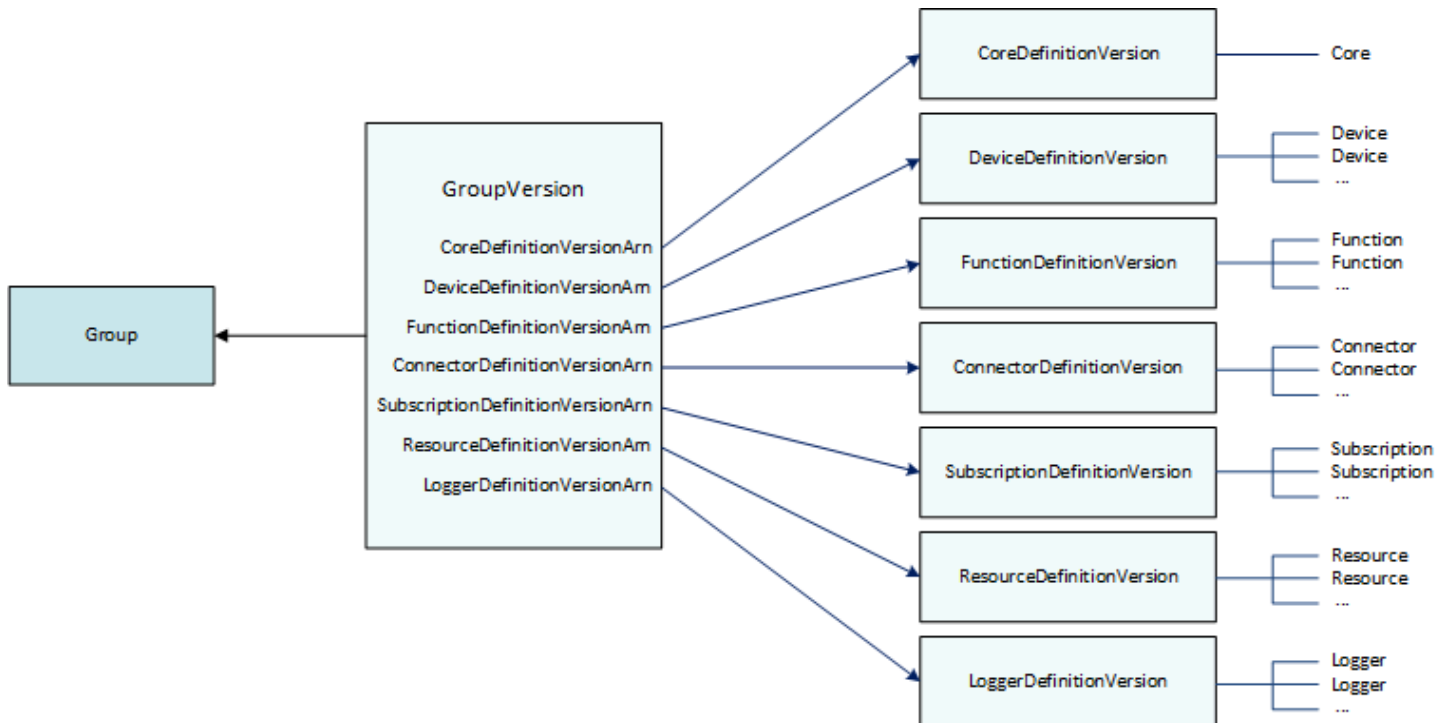
Gruppen

In der AWS IoT Greengrass API besteht das Group Objekt der obersten Ebene aus Metadaten und einer Liste von GroupVersion Objekten. GroupVersionObjekte sind mit einer Group BY-ID verknüpft.



Gruppenversionen

GroupVersion-Objekte definieren die Gruppenmitgliedschaft. Jede GroupVersion verweist auf eine CoreDefinitionVersion und andere Komponentenversionen nach ARN. Diese Referenzen bestimmen, welche Entitäten in die Gruppe aufgenommen werden sollen.



Um beispielsweise drei Lambda-Funktionen, ein Gerät und zwei Abonnements in die Gruppe aufzunehmen, lauten die GroupVersion Verweise:

- Die CoreDefinitionVersion, die den erforderlichen Core enthält.
- Die FunctionDefinitionVersion, die die drei Funktionen enthält.
- Das DeviceDefinitionVersion, das das Client-Gerät enthält.
- Die SubscriptionDefinitionVersion, die die beiden Abonnements enthält.

Die für ein Core-Gerät bereitgestellte GroupVersion bestimmt die Entitäten, die in der lokalen Umgebung verfügbar sind, und wie sie interagieren können.

Gruppenkomponenten

Komponenten, die Sie Gruppen hinzufügen, haben eine dreistufige Hierarchie:

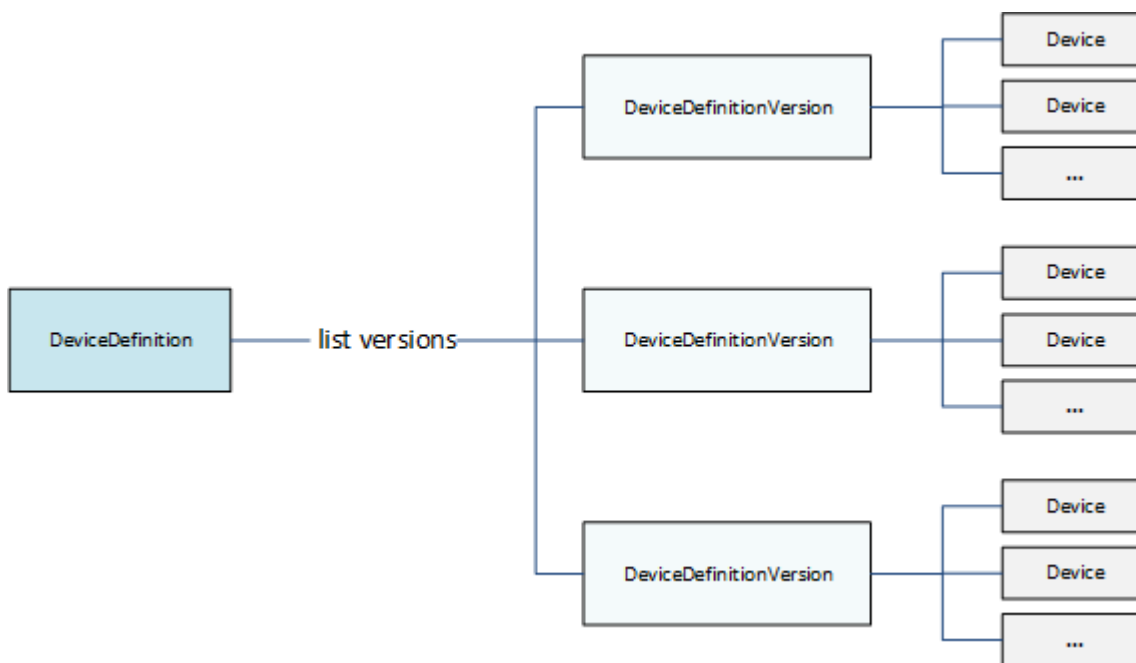
- Eine Definition, die auf eine Liste von DefinitionVersionObjekten eines bestimmten Typs verweist. Beispielsweise verweist eine DeviceDefinition auf eine Liste von DeviceDefinitionVersion-Objekten.
- A DefinitionVersion, die eine Menge von Entitäten eines bestimmten Typs enthält. Eine DeviceDefinitionVersion enthält beispielsweise eine Liste von Device-Objekten.

- Einzelne Entitäten, die ihre Eigenschaften und Verhalten definieren. Beispielsweise Device definiert a den ARN des entsprechenden Client-Geräts in der AWS IoT Registrierung, den ARN seines Gerätezertifikats und ob sein lokaler Shadow automatisch mit der Cloud synchronisiert wird.

Sie können einer Gruppe die folgenden Arten von Entitäten hinzufügen:

- [Konnektor](#)
- [Core](#)
- [Gerät](#)
- [Funktion](#)
- [Logger](#)
- [Ressource](#)
- [Abonnement](#)

Das folgende Beispiel einer DeviceDefinition verweist auf drei DeviceDefinitionVersion-Objekte, die jeweils mehrere Device-Objekte enthalten. Es wird nur jeweils eine DeviceDefinitionVersion in einer Gruppe verwendet.



Aktualisieren von Gruppen

In der AWS IoT Greengrass API verwenden Sie Versionen, um die Konfiguration einer Gruppe zu aktualisieren. Versionen sind unveränderlich. Um Gruppenkomponenten hinzuzufügen, zu entfernen

oder zu ändern, müssen Sie DefinitionVersionObjekte erstellen, die neue oder aktualisierte Entitäten enthalten.

Sie können neue DefinitionVersionsObjekte neuen oder vorhandenen Definitionsobjekten zuordnen. Sie können beispielsweise die CreateFunctionDefinition-Aktion verwenden, um eine FunctionDefinition zu erstellen, die die FunctionDefinitionVersion als anfängliche Version enthält. Sie können auch die CreateFunctionDefinitionVersion-Aktion verwenden und auf eine vorhandene FunctionDefinition verweisen.

Nachdem Sie Ihre Gruppenkomponenten erstellt haben, erstellen Sie eineGroupVersion, die alle DefinitionVersionObjekte enthält, die Sie in die Gruppe aufnehmen möchten. Stellen Sie anschließend die GroupVersion bereit.

Um eine GroupVersion bereitzustellen, muss sie auf eine CoreDefinitionVersion verweisen, die genau einen Core enthält. Alle referenzierten Entitäten müssen Mitglieder der Gruppe sein. Außerdem muss Ihnen in dem Land, AWS-Konto in AWS-Region dem Sie die bereitstellen, [eine Greengrass-Servicerolle](#) zugeordnet sein. GroupVersion

 Note

Die Update-Aktionen in der API werden verwendet, um den Namen einer Group oder eines Komponentendefinitionsobjekts zu ändern.

Entitäten, die auf Ressourcen verweisen AWS , werden aktualisiert

Greengrass Lambda-Funktionen und [geheime Ressourcen](#) definieren Greengrass-spezifische Eigenschaften und verweisen auch auf entsprechende Ressourcen. AWS Um diese Entitäten zu aktualisieren, können Sie anstelle Ihrer Greengrass-Objekte Änderungen an der entsprechenden AWS Ressource vornehmen. Lambda-Funktionen verweisen beispielsweise auf eine Funktion in AWS Lambda und definieren auch den Lebenszyklus und andere Eigenschaften, die für die Greengrass-Gruppe spezifisch sind.

- Um den Lambda-Funktionscode oder die Paketabhängigkeiten zu aktualisieren, nehmen Sie Ihre Änderungen in AWS Lambda vor. Bei der nächsten Gruppenbereitstellung werden diese Änderungen aus Ihrer lokalen Umgebung abgerufen AWS Lambda und in diese kopiert.
- Um [Greengrass-spezifische Eigenschaften](#) zu aktualisieren, erstellen Sie eine FunctionDefinitionVersion, der die aktualisierten Function-Eigenschaften enthält.

 Note

Lambda-Funktionen von Greengrass können auf eine Lambda-Funktion mit dem Alias ARN oder der Version ARN verweisen. Wenn Sie auf den Alias-ARN verweisen (empfohlen), müssen Sie Ihre `FunctionDefinitionVersion` (oder `SubscriptionDefinitionVersion`) nicht aktualisieren, wenn Sie eine neue Funktionsversion in AWS Lambda veröffentlichen. Weitere Informationen finden Sie unter [the section called “Referenzieren von Funktionen nach Alias oder Version”](#).

Weitere Informationen finden Sie auch unter

- [the section called “Abrufen von Bereitstellungsbenachrichtigungen”](#)
- [the section called “Zurücksetzen von Bereitstellungen”](#)
- [the section called “Erstellen von Sammelbereitstellungen”](#)
- [Fehlerbehebung bei Bereitstellungsproblemen](#)
- [AWS IoT Greengrass Version 1 API Reference](#)
- [AWS IoT GreengrassBefehle in der Befehlsreferenz AWS CLI](#)

Abrufen von Bereitstellungsbenachrichtigungen

Mit den EventBridge Amazon-Veranstaltungsregeln erhalten Sie Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen. EventBridge liefert nahezu in Echtzeit einen Stream von Systemereignissen, der Änderungen an AWS Ressourcen beschreibt. AWS IoT Greengrass sendet diese EventBridge Ereignisse mindestens einmal an. Das bedeutet, dass AWS IoT Greengrass möglicherweise mehrere Kopien eines bestimmten Ereignisses gesendet werden, um die Zustellung sicherzustellen. Außerdem empfangen Ihre Ereignis-Listener die Ereignisse möglicherweise nicht in der Reihenfolge, in der die Ereignisse aufgetreten sind.

 Note

Amazon EventBridge ist ein Event-Bus-Service, mit dem Sie Ihre Anwendungen mit Daten aus einer Vielzahl von Quellen verbinden können, wie z. B. [Greengrass-Core-Geräten](#) und

Bereitstellungsbenachrichtigungen. Weitere Informationen finden Sie unter [Was ist Amazon EventBridge?](#) im EventBridge Amazon-Benutzerhandbuch.

AWS IoT Greengrass gibt ein Ereignis aus, wenn sich der Status von Gruppenbereitstellungen ändert. Sie können eine EventBridge Regel erstellen, die für alle von Ihnen angegebenen Zustandsübergänge oder Zustandsübergänge gilt. Wenn eine Bereitstellung in einen Zustand übergeht, der eine Regel initiiert, EventBridge ruft sie die in der Regel definierten Zielaktionen auf. Auf diese Weise können Sie Benachrichtigungen senden, Ereignisinformationen erfassen, Korrekturmaßnahmen ergreifen oder andere Ereignisse als Reaktion auf eine Statusänderung initiieren. Sie können beispielsweise Regeln für die folgenden Anwendungsfälle erstellen:

- Initiiert Operationen nach der Bereitstellung, z. B. das Herunterladen von Ressourcen und die Benachrichtigung des Personals.
- Senden Sie Benachrichtigungen bei erfolgreicher oder fehlgeschlagener Bereitstellung.
- Veröffentlichen Sie benutzerdefinierte Metriken zu Bereitstellungsereignissen.

AWS IoT Greengrass löst ein Ereignis aus, wenn eine Bereitstellung in die folgenden Zustände übergeht: `Building`, `InProgress`, `Success` und `Failure`

Note

Die Überwachung des Status einer [Sammelbereitstellungsoperation](#) wird derzeit nicht unterstützt. Gibt jedoch Zustandsänderungsereignisse für einzelne Gruppenbereitstellungen aus, die Teil einer AWS IoT Greengrass Massenbereitstellung sind.

Änderungsereignis für den Gruppenbereitstellungsstatus

Das [Ereignis](#) für eine Änderung des Bereitstellungsstatus verwendet das folgende Format:

```
{
  "version": "0",
  "id": " cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
  "detail-type": "Greengrass Deployment Status Change",
  "source": "aws.greengrass",
  "account": "123456789012",
```

```
{
  "time": "2018-03-22T00:38:11Z",
  "region": "us-west-2",
  "resources": [],
  "detail": {
    "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
    "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
    "deployment-type": "NewDeployment|Redeployment|ResetDeployment|
ForceResetDeployment",
    "status": "Building|InProgress|Success|Failure"
  }
}
```

Sie können Regeln erstellen, die für mindestens eine Gruppe gelten. Sie können Regeln nach mindestens einem der folgenden Bereitstellungstypen und Bereitstellungsstatus filtern:

Bereitstellungstypen

- **NewDeployment.** Die erste Bereitstellung einer Gruppenversion.
- **ReDeployment.** Eine erneute Bereitstellung einer Gruppenversion.
- **ResetDeployment.** Löscht Bereitstellungsinformationen, die im AWS Cloud und auf dem AWS IoT Greengrass Core gespeichert sind. Weitere Informationen finden Sie unter [the section called “Zurücksetzen von Bereitstellungen”](#).
- **ForceResetDeployment.** Löscht die in der gespeicherten Bereitstellungsinformationen AWS Cloud und meldet den Erfolg, ohne auf die Antwort des Kerns zu warten. Löscht außerdem Bereitstellungsinformationen, die auf dem Core gespeichert sind, wenn der Core verbunden ist oder wenn er das nächste Mal eine Verbindung herstellt.

Bereitstellungsstatus

- **Building.** AWS IoT Greengrass validiert die Gruppenkonfiguration und erstellt Einrichtungsartefakte.
- **InProgress.** Die Bereitstellung im AWS IoT Greengrass Kern ist im Gange.
- **Success.** Die Bereitstellung war erfolgreich.
- **Failure.** Die Bereitstellung ist fehlgeschlagen.

Es ist möglich, dass Ereignisse doppelt sind oder nicht in der richtigen Reihenfolge werden. Um die Reihenfolge der Ereignisse zu bestimmen, verwenden Sie die `time`-Eigenschaft.

 Note

AWS IoT Greengrass verwendet die `resources` Eigenschaft nicht, daher ist sie immer leer.

Voraussetzungen für die Erstellung von EventBridge Regeln

Bevor Sie eine EventBridge Regel für erstellen AWS IoT Greengrass, gehen Sie wie folgt vor:

- Machen Sie sich mit Ereignissen, Regeln und Zielen in vertraut EventBridge.
- Erstellen und konfigurieren Sie die Ziele, die durch Ihre EventBridge Regeln aufgerufen werden. Regeln können viele Arten von Zielen aufrufen, einschließlich:
 - Amazon-Simple-Notification-Service (Amazon-SNS)
 - AWS Lambda Funktionen
 - Amazon Kinesis Video Streams
 - Amazon-Simple-Queue-Service-(Amazon-SQS)-Warteschlangen

Weitere Informationen finden Sie unter [Was ist Amazon EventBridge?](#) und [Erste Schritte mit Amazon EventBridge](#) im EventBridge Amazon-Benutzerhandbuch.

Konfigurieren von Bereitstellungsbenachrichtigungen (Konsole)

Gehen Sie wie folgt vor, um eine EventBridge Regel zu erstellen, die ein Amazon SNS SNS-Thema veröffentlicht, wenn sich der Bereitstellungsstatus für eine Gruppe ändert. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren. Weitere Informationen finden Sie im EventBridge Amazon-Benutzerhandbuch unter [Erstellen einer EventBridge Regel, die bei einem Ereignis von einer AWS Ressource ausgelöst wird](#).

1. Öffnen Sie die [EventBridgeAmazon-Konsole](#).
2. Wählen Sie im Navigationsbereich Regeln aus.
3. Wählen Sie Regel erstellen aus.
4. Geben Sie einen Namen und eine Beschreibung für die Regel ein.

Eine Regel darf nicht denselben Namen wie eine andere Regel in derselben Region und auf demselben Event Bus haben.

5. Wählen Sie als Event bus (Event Bus) den Event Bus aus, den Sie dieser Regel zuordnen möchten. Wenn Sie möchten, dass diese Regel mit Ereignissen aus Ihrem eigenen Konto übereinstimmt, wählen Sie AWS -Standard-Event-Bus aus. Wenn ein AWS Service in Ihrem Konto ein Ereignis ausgibt, wird es immer an den Standard-Event-Bus Ihres Kontos weitergeleitet.
6. Bei Regeltyp wählen Sie Regel mit einem Ereignismuster aus.
7. Wählen Sie Weiter aus.
8. Als Event source (Ereignisquelle) wählen Sie AWS -Services aus.
9. Wählen Sie für Event-Pattern die Option AWS Services aus.
10. Wählen Sie für den AWS Service Greengrass.
11. Wählen Sie für Ereignistyp die Option Greengrass Deployment Status Change (Änderung des Status der Greengrass-Bereitstellung).

 Note

Der AWS API-Aufruf über den CloudTrail Ereignistyp basiert auf der AWS IoT Greengrass Integration mit AWS CloudTrail. Sie können diese Option verwenden, um Regeln zu erstellen, die durch Lese- oder Schreibaufrufe an die AWS IoT Greengrass API initiiert werden. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass API-Aufrufe protokollieren mit AWS CloudTrail”](#).

12. Wählen Sie die Bereitstellungsstatus aus, die eine Benachrichtigung auslösen.
 - Um Benachrichtigungen für alle Statusänderungsereignisse zu erhalten, wählen Sie Any state (Beliebiger Status).
 - Um Benachrichtigungen nur für bestimmte Statusänderungsereignisse zu erhalten, wählen Sie Specific state(s) (Spezifische(r) Status) und dann die Zielstatus aus.
13. Wählen Sie die Bereitstellungstypen aus, die eine Benachrichtigung auslösen.
 - Um Benachrichtigungen für alle Bereitstellungstypen zu erhalten, wählen Sie Any state (Beliebiger Status).
 - Um Benachrichtigungen nur für bestimmte Bereitstellungstypen zu erhalten, wählen Sie Specific state(s) (Spezifische(r) Status) und dann die Zielbereitstellungstypen aus.
14. Wählen Sie Weiter aus.
15. Bei Zieltypen wählen Sie AWS -Service aus.

16. Für Wählen Sie ein Ziel aus konfigurieren Sie Ihr Ziel. In diesem Beispiel wird ein Amazon SNS SNS-Thema verwendet, Sie können jedoch auch andere Zieltypen für das Senden von Benachrichtigungen konfigurieren.
 - a. Wählen Sie in Target (Ziel) die Option SNS topic (SNS-Thema) aus.
 - b. Wählen Sie unter Thema das Zielthema aus.
 - c. Wählen Sie Weiter aus.
17. Definieren Sie unter Tags Tags für die Regel oder lassen Sie die Felder leer.
18. Wählen Sie Weiter aus.
19. Überprüfen Sie die Details der Regel und wählen Sie dann Regel erstellen aus.

Konfigurieren von Bereitstellungsbenachrichtigungen (CLI)

Gehen Sie wie folgt vor, um eine EventBridge Regel zu erstellen, die ein Amazon SNS SNS-Thema veröffentlicht, wenn sich der Bereitstellungsstatus für eine Gruppe ändert. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren.

1. Erstellen Sie die Regel.
 - *group-id* Ersetzen Sie es durch die ID Ihrer AWS IoT Greengrass Gruppe.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":  
    [\"group-id\"]}}"
```

Eigenschaften, die aus dem Muster weggelassen werden, werden ignoriert.

2. Fügen Sie das Thema als Regelziel hinzu.
 - *topic-arn* Ersetzen Sie es durch den ARN Ihres Amazon SNS SNS-Themas.

```
aws events put-targets \  
  --rule TestRule \  
  --targets "Id"="1", "Arn"="topic-arn"
```

 Note

Damit Amazon EventBridge Ihr Zielthema aufrufen kann, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unter [Amazon SNS SNS-Berechtigungen](#) im EventBridge Amazon-Benutzerhandbuch.

Weitere Informationen finden Sie unter [Ereignisse und Ereignismuster EventBridge im EventBridge Amazon-Benutzerhandbuch](#).

Konfigurieren von Bereitstellungsbenachrichtigungen (CloudFormation)

Verwenden Sie CloudFormation Vorlagen, um EventBridge Regeln zu erstellen, die Benachrichtigungen über Statusänderungen für Ihre Greengrass-Gruppenbereitstellungen senden. Weitere Informationen finden Sie in der [Referenz zum EventBridge Amazon-Ressourcentyp](#) im AWS CloudFormation Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- [Stellen Sie AWS IoT Greengrass Gruppen bereit](#)
- [Was ist Amazon EventBridge?](#) im EventBridge Amazon-Benutzerhandbuch

Zurücksetzen von Bereitstellungen

Diese Funktion ist für AWS IoT Greengrass Core v1.1 und höher verfügbar.

Sie können die Bereitstellungen einer Gruppe zurücksetzen auf:

- Löschen Sie die Gruppe, z. B. wenn Sie den Kern der Gruppe in eine andere Gruppe verschieben möchten oder wenn für den Kern der Gruppe ein neues Image erstellt wurde. Bevor Sie eine Gruppe löschen, müssen Sie die Bereitstellungen der Gruppe zurücksetzen, um den Core mit einer anderen Greengrass-Gruppe zu verwenden.
- Der Core der Gruppe soll in eine andere Gruppe verschoben werden.
- Die Gruppe soll wieder in den Zustand vor der Bereitstellung versetzt werden.
- Die Bereitstellungsconfiguration soll vom Core-Gerät entfernt werden.
- Sensible Daten sollen vom Core-Gerät oder aus der Cloud gelöscht werden.

- Eine neue Gruppenkonfiguration soll auf einem Core bereitgestellt werden, ohne dass der Core durch einen anderen in der aktuellen Gruppe ersetzt wird.

Note

Die Funktionalität zum Zurücksetzen von Bereitstellungen ist in AWS IoT Greengrass Core Software v1.0.0 nicht verfügbar. Sie können keine Gruppe löschen, die mit v1.0.0 bereitgestellt wurde.

Die Operation zum Zurücksetzen von Bereitstellungen bereinigt zunächst alle Bereitstellungsinformationen, die in der Cloud für eine bestimmte Gruppe gespeichert sind. Anschließend weist es das Kerngerät der Gruppe an, auch alle Informationen zur Bereitstellung zu bereinigen (Lambda-Funktionen, Benutzerprotokolle, Schattendatenbank und Serverzertifikat, aber nicht die benutzerdefinierten `config.json` oder die Greengrass-Core-Zertifikate). Die Bereitstellungen einer Gruppe können nicht zurückgesetzt werden, wenn für diese Gruppe aktuell eine Bereitstellung mit dem Status `In Progress` oder `Building` vorhanden ist.

Setzen Sie Bereitstellungen von der Konsole aus zurück AWS IoT

Sie können Gruppenbereitstellungen auf der Gruppenkonfigurationsseite in der AWS IoT Konsole zurücksetzen.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Registerkarte Bereitstellungen die Option Bereitstellungen zurücksetzen aus.
4. Geben Sie im Dialogfeld Bereitstellungen für diese Greengrass-Gruppe zurücksetzen den Text ein, **confirm** um zuzustimmen, und wählen Sie Bereitstellung zurücksetzen aus.

Setzen Sie Bereitstellungen mit der API zurück AWS IoT Greengrass

Sie können die `ResetDeployments` Aktion in der AWS CLI AWS IoT Greengrass API oder im AWS SDK verwenden, um Bereitstellungen zurückzusetzen. Die Beispiele in diesem Thema verwenden die CLI.

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

Argumente für den CLI-Befehl **reset-deployments**:

--group-id

Die Gruppen-ID. Verwenden Sie den `list-groups`-Befehl, um diesen Wert abzurufen.

--force

Optional. Verwenden Sie diesen Parameter, wenn das Core-Gerät der Gruppe verloren, gestohlen oder zerstört wurde. Diese Option bewirkt, dass während des Zurücksetzens der Bereitstellungen eine Erfolgsmeldung ausgegeben wird, sobald alle Bereitstellungsinformationen in der Cloud gelöscht wurden, ohne auf die Antwort des Core-Geräts zu warten. Wenn das Core-Gerät jedoch aktiv ist oder wird, führt es auch Bereinigungsoperationen durch.

Die Ausgabe des CLI-Befehls `reset-deployments` sieht wie folgt aus:

```
{
  "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
  "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

Sie können den Status der Bereitstellungsrücksetzung mit dem CLI-Befehl `get-deployment-status` prüfen.

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

Argumente für den CLI-Befehl **get-deployment-status**:

--deployment-id

Die Bereitstellungs-ID.

--group-id

Die Gruppen-ID.

Die Ausgabe des CLI-Befehls `get-deployment-status` sieht wie folgt aus:


```
{
  "DeploymentStatus": "Success",
  "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

Der Wert von `DeploymentStatus` wird auf `Building` gesetzt, wenn das Zurücksetzen der Bereitstellungen vorbereitet wird. Wenn die Reset-Bereitstellung bereit ist, der AWS IoT Greengrass Core die Reset-Bereitstellung jedoch noch nicht übernommen hat, `DeploymentStatus` ist `InProgress` dies der Fall.

Wenn das Zurücksetzen fehlschlägt, werden Fehlerinformationen in der Antwort zurückgegeben.

Weitere Informationen finden Sie auch unter

- [Stellen Sie AWS IoT Greengrass Gruppen bereit](#)
- [ResetDeployments](#) in der AWS IoT Greengrass Version 1 API-Referenz
- [GetDeploymentStatus](#) in der AWS IoT Greengrass Version 1 API-Referenz

Erstellen von Sammelbereitstellungen für Gruppen

Sie können einfache API-Aufrufe verwenden, um eine große Anzahl von Greengrass-Gruppen auf einmal bereitzustellen. Diese Bereitstellungen werden mit einer adaptiven Rate ausgelöst, die eine feste Obergrenze hat.

In diesem Tutorial wird beschrieben, wie Sie die AWS CLI zum Erstellen und Überwachen einer Massengruppenbereitstellung in verwenden AWS IoT Greengrass. Das Beispiel für die Sammelbereitstellung in diesem Tutorial enthält mehrere Gruppen. Sie können das Beispiel verwenden, um in Ihrer Implementierung so viele Gruppen hinzuzufügen, wie Sie benötigen.

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Erstellen und Hochladen der Eingabedatei für die Sammelbereitstellung](#)
2. [Erstellen und konfigurieren Sie eine IAM-Ausführungsrolle für Massenbereitstellungen](#)
3. [Ermöglichen des Zugriff auf Ihren S3-Bucket für Ihre Ausführungsrolle](#)
4. [Bereitstellen der Gruppen](#)
5. [Testen der Bereitstellung](#)

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine oder mehrere bereitstellbare Greengrass-Gruppen. Weitere Informationen zum Erstellen von AWS IoT Greengrass -Gruppen und -Kernen finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#).
- Das AWS CLI ist auf Ihrem Computer installiert und konfiguriert. Weitere Informationen finden Sie im [AWS CLI -Benutzerhandbuch](#).
- Ein S3-Bucket, der im selben Format AWS-Region wie erstellt wurde AWS IoT Greengrass. Weitere Informationen finden Sie unter [Erstellen und Konfigurieren eines S3-Buckets](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Note

Derzeit werden SSE KMS-aktivierte Buckets nicht unterstützt.

Schritt 1: Erstellen und Hochladen der Eingabedatei für die Sammelbereitstellung

In diesem Schritt erstellen Sie eine Eingabedatei für die Bereitstellung und laden sie in Ihren Amazon S3 S3-Bucket hoch. Diese Datei ist eine serialisierte, zeilengetrennte JSON-Datei, die Informationen zu jeder Gruppe in Ihrer Massenbereitstellung enthält. AWS IoT Greengrass verwendet diese Informationen, um jede Gruppe in Ihrem Namen bereitzustellen, wenn Sie Ihre Massengruppenbereitstellung initialisieren.

1. Führen Sie den folgenden Befehl aus, um die groupId für jede Gruppe zu erhalten, die Sie bereitstellen möchten. Sie geben die groupId in Ihrer Eingabedatei für die Sammelbereitstellung ein, damit AWS IoT Greengrass jede bereitzustellende Gruppe identifizieren kann.

Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

```
aws greengrass list-groups
```

Die Antwort enthält Informationen zu jeder Gruppe in Ihrem AWS IoT Greengrass Konto:

```
{
  "Groups": [
    {
      "Name": "string",
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

Führen Sie den folgenden Befehl aus, um die `groupVersionId` jeder Gruppe zu erhalten, die Sie bereitstellen möchten.

```
list-group-versions --group-id groupId
```

Die Antwort enthält Informationen über alle Versionen in der Gruppe. Notieren Sie sich den `Version` Wert für die Gruppenversion, die Sie verwenden möchten.

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
```

```
    "Version": "string",
    "CreationTimestamp": "string"
  },
  "NextToken": "string"
}
```

2. Erstellen Sie in Ihrem Computerterminal oder Editor Ihrer Wahl eine Datei *MyBulkDeploymentInputFile* anhand des folgenden Beispiels. Diese Datei enthält Informationen zu jeder AWS IoT Greengrass Gruppe, die in eine Massenbereitstellung aufgenommen werden soll. Obwohl in diesem Beispiel mehrere Gruppen definiert sind, kann Ihre Datei für dieses Tutorial nur eine enthalten.

 Note

Die Größe dieser Datei muss kleiner als 100 MB sein.

```
{ "GroupId": "groupId1", "GroupVersionId": "groupVersionId1",
  "DeploymentType": "NewDeployment" }
{ "GroupId": "groupId2", "GroupVersionId": "groupVersionId2",
  "DeploymentType": "NewDeployment" }
{ "GroupId": "groupId3", "GroupVersionId": "groupVersionId3",
  "DeploymentType": "NewDeployment" }
...
```

Jeder Datensatz (oder Zeile) enthält ein Gruppenobjekt. Jedes Gruppenobjekt enthält seine entsprechende GroupId und GroupVersionId und einen DeploymentType. AWS IoT Greengrass unterstützt derzeit nur Typen von NewDeployment Massenbereitstellungen.

Speichern und schließen Sie Ihre Datei. Notieren Sie sich den Speicherort der Datei.

3. Verwenden Sie den folgenden Befehl in Ihrem Terminal, um Ihre Eingabedatei in Ihren Amazon S3 S3-Bucket hochzuladen. Ersetzen Sie den Dateipfad durch den Speicherort und den Namen Ihrer Datei. Weitere Informationen finden Sie unter [Hinzufügen eines Objekts zu einem Bucket](#).

```
aws s3 cp path/MyBulkDeploymentInputFile s3://amzn-s3-demo-bucket/
```

Schritt 2: Erstellen und Konfigurieren einer IAM-Ausführungsrolle

In diesem Schritt verwenden Sie die IAM-Konsole, um eine eigenständige Ausführungsrolle zu erstellen. Anschließend richten Sie eine Vertrauensbeziehung zwischen der Rolle ein AWS IoT Greengrass und stellen sicher, dass Ihr IAM-Benutzer über `PassRole` Berechtigungen für Ihre Ausführungsrolle verfügt. Auf diese Weise können AWS IoT Greengrass Sie Ihre Ausführungsrolle übernehmen und die Bereitstellungen in Ihrem Namen erstellen.

1. Verwenden Sie die folgende Richtlinie, um eine Ausführungsrolle zu erstellen. Dieses Richtliniendokument ermöglicht es AWS IoT Greengrass, auf Ihre Eingabedatei für die Sammelbereitstellung zuzugreifen, wenn es jede Bereitstellung in Ihrem Namen erstellt.

Weitere Informationen zum Erstellen einer IAM-Rolle und zum Delegieren von Berechtigungen finden Sie unter [Erstellen von IAM-Rollen](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "greengrass:CreateDeployment",
      "Resource": [
        "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId1",
        "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId2",
        "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId3"
      ]
    }
  ]
}
```

Note

Diese Richtlinie muss für jede Gruppe oder Gruppenversion in Ihrer Eingabedatei eine Ressource für die Sammelbereitstellung enthalten, die von AWS IoT Greengrass

bereitzustellen ist. Um den Zugriff auf alle Gruppen zu ermöglichen, geben Sie für Resource ein Sternchen an:

```
"Resource": ["*"]
```

2. Ändern Sie die Vertrauensbeziehung für Ihre Ausführungsrolle, um AWS IoT Greengrass einzuschließen. Dadurch kann AWS IoT Greengrass Ihre Ausführungsrolle und die damit verbundenen Berechtigungen verwenden. Weitere Informationen finden Sie unter [Bearbeiten der Vertrauensstellung für eine vorhandene Rolle](#).

Wir empfehlen Ihnen, auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufzunehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie [Serviceübergreifende Confused-Deputy-Prävention](#).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:us-east-1:123456789012:*"
        }
      }
    }
  ]
}
```

```
}
```

3. Erteilen Sie Ihrem PassRole IAM-Benutzer IAM-Berechtigungen für Ihre Ausführungsrolle. Dieser IAM-Benutzer wird verwendet, um die Massenbereitstellung zu initiieren. PassRoleBerechtigungen ermöglichen es Ihrem IAM-Benutzer, Ihre Ausführungsrolle zur Verwendung an ihn zu AWS IoT Greengrass übergeben. Weitere Informationen finden Sie unter [Einem Benutzer Berechtigungen zur Übergabe einer Rolle an einen AWS Dienst](#) erteilen.

Verwenden Sie das folgende Beispiel, um die Ihrer Ausführungsrolle zugeordnete IAM-Richtlinie zu aktualisieren. Ändern Sie dieses Beispiel, falls erforderlich.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1508193814000",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:user/executionRoleArn"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "greengrass.amazonaws.com"
        }
      }
    }
  ]
}
```

Schritt 3: Ermöglichen des Zugriffs auf Ihren S3-Bucket für Ihre Ausführungsrolle

Um Ihre Massenbereitstellung zu starten, muss Ihre Ausführungsrolle in der Lage sein, Ihre Eingabedatei für die Massenbereitstellung aus Ihrem Amazon S3 S3-Bucket zu lesen. Hängen Sie

die folgende Beispielrichtlinie an Ihren Amazon S3 S3-Bucket an, damit Ihre Ausführungsrolle auf dessen `GetObject` Berechtigungen zugreifen kann.

Weitere Informationen finden Sie unter [Wie füge ich eine S3-Bucket-Richtlinie hinzu?](#)

JSON

```
{
  "Version": "2012-10-17",
  "Id": "examplePolicy",
  "Statement": [
    {
      "Sid": "Stmt1535408982966",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "executionRoleArn"
        ]
      },
      "Action": "s3:GetObject",
      "Resource":
        "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"
    }
  ]
}
```

Sie können den folgenden Befehl in Ihrem Terminal verwenden, um die Richtlinien Ihres Buckets zu überprüfen:

```
aws s3api get-bucket-policy --bucket amzn-s3-demo-bucket
```

Note

Sie können Ihre Ausführungsrolle direkt ändern, um ihr stattdessen die `GetObject` Berechtigungen Ihres Amazon S3 S3-Buckets zu gewähren. Fügen Sie dazu Ihrer Ausführungsrolle die folgende Beispielrichtlinie an.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"
    }
  ]
}
```

Schritt 4: Bereitstellen der Gruppen

In diesem Schritt starten Sie eine Sammelbereitstellung für alle Gruppenversionen, die in Ihrer Eingabedatei für die Sammelbereitstellung konfiguriert sind. Die Bereitstellungsaktion für jede Ihrer Gruppenversionen ist vom Typ `NewDeploymentType`.

Note

Sie können `StartBulkDeployment` nicht aufrufen, während eine andere Sammelbereitstellung aus demselben Konto noch läuft. Die Anforderung wurde abgelehnt.

1. Verwenden Sie den folgenden Befehl, um die Sammelbereitstellung zu starten.

Wir empfehlen, dass Sie ein `X-Amzn-Client-Token` in jeder `StartBulkDeployment` Anforderung einschließen. Diese Anforderungen sind in Bezug auf das Token und die Anfrageparameter idempotent. Dieses Token kann eine beliebige eindeutige Zeichenfolge, bei der die Groß- und Kleinschreibung zu beachten ist, mit bis zu 64 ASCII-Zeichen sein.

```
aws greengrass start-bulk-deployment --cli-input-json "{
  \"InputFileUri\": \"URI of file in S3 bucket\",
```

```
"ExecutionRoleArn": "ARN of execution role",  
"AmznClientToken": "your Amazon client token"  
}"
```

Der Befehl sollte zu einem erfolgreichen Statuscode 200 führen, zusammen mit der folgenden Antwort:

```
{  
  "bulkDeploymentId": UUID  
}
```

Notieren Sie sich die ID der Sammelbereitstellung. Sie kann verwendet werden, um den Status Ihrer Sammelbereitstellung zu überprüfen.

 Note

Obwohl Massenbereitstellungsvorgänge derzeit nicht unterstützt werden, können Sie EventBridge Amazon-Veranstaltungsregeln erstellen, um Benachrichtigungen über Änderungen des Bereitstellungsstatus für einzelne Gruppen zu erhalten. Weitere Informationen finden Sie unter [the section called “Abrufen von Bereitstellungsbenachrichtigungen”](#).

2. Verwenden Sie den folgenden Befehl, um den Status Ihrer Sammelbereitstellung zu überprüfen:

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

Der Befehl sollte einen erfolgreichen Statuscode 200 zurückgeben, zusätzlich zu einer JSON-Nutzlast aus Informationen:

```
{  
  "BulkDeploymentStatus": Running,  
  "Statistics": {  
    "RecordsProcessed": integer,  
    "InvalidInputRecords": integer,  
    "RetryAttempts": integer  
  },  
}
```

```
"CreatedAt": "string",
"ErrorMessage": "string",
"ErrorDetails": [
  {
    "DetailedErrorCode": "string",
    "DetailedErrorMessage": "string"
  }
]
```

`BulkDeploymentStatus` enthält den aktuellen Status der Sammelausführung. Die Ausführung kann einen von sechs verschiedenen Statusarten haben:

- **Initializing.** Die Anfrage zur Massenbereitstellung wurde empfangen, und der Start der Ausführung wird gerade vorbereitet.
- **Running.** Die Ausführung der Massenbereitstellung wurde gestartet.
- **Completed.** Die Ausführung der Massenbereitstellung hat die Verarbeitung aller Datensätze abgeschlossen.
- **Stopping.** Die Ausführung der Massenbereitstellung hat einen Befehl zum Beenden erhalten und wird in Kürze beendet. Sie können keine neue Sammelbereitstellung starten, solange sich eine frühere Bereitstellung noch nicht im Status `Stopping` befindet.
- **Stopped.** Die Ausführung der Massenbereitstellung wurde manuell gestoppt.
- **Failed.** Bei der Ausführung der Massenbereitstellung ist ein Fehler aufgetreten und sie wurde beendet. Fehlerdetails finden Sie im `ErrorDetails`-Feld.

Die JSON-Nutzlast enthält auch statistische Informationen über den Fortschritt der Sammelbereitstellung. Anhand dieser Informationen können Sie feststellen, wie viele Gruppen verarbeitet wurden und wie viele fehlgeschlagen sind. Die statistischen Informationen umfassen:

- **RecordsProcessed:** Die Anzahl der Gruppensätze, die versucht wurden.
- **InvalidInputRecords:** Die Gesamtzahl der Datensätze, die einen nicht wiederholbaren Fehler (Non-Retryable Error) zurückgegeben haben. Dies kann beispielsweise der Fall sein, wenn ein Gruppendatensatz aus der Eingabedatei ein ungültiges Format aufweist oder eine nicht vorhandene Gruppenversion angibt, oder wenn die Ausführung keine Berechtigung zum Bereitstellen einer Gruppen- oder Gruppenversion erteilt.

- **RetryAttempts:** Die Anzahl der Bereitstellungsversuche, die einen wiederholbaren Fehler (Retryable Error) zurückgegeben haben. Beispielsweise wird ein Wiederholungsversuch ausgelöst, wenn der Versuch, eine Gruppe bereitzustellen, einen Ablehnungsfehler (Throttling Error) zurückgibt. Eine Gruppenbereitstellung kann bis zu fünfmal wiederholt werden.

Im Falle eines Ausführungsfehlers bei der Sammelbereitstellung beinhaltet diese Nutzlast auch einen Abschnitt `ErrorDetails`, der zur Fehlerbehebung verwendet werden kann. Hierin sind Informationen über die Ursache des Ausführungsfehlers enthalten.

Sie können den Status der Sammelbereitstellung regelmäßig überprüfen, um sicherzustellen, dass sie wie erwartet verläuft. Nachdem die Bereitstellung abgeschlossen ist, sollte `RecordsProcessed` gleich der Anzahl der Bereitstellungsgruppen in Ihrer Eingabedatei für die Sammelbereitstellung sein. Dies zeigt an, dass jeder Datensatz bearbeitet wurde.

Schritt 5: Testen der Bereitstellung

Verwenden Sie den `ListBulkDeployments`-Befehl, um die ID Ihrer Sammelbereitstellung zu finden.

```
aws greengrass list-bulk-deployments
```

Dieser Befehl gibt eine Liste aller Ihrer Sammelbereitstellung von der neuesten bis zur ältesten zurück, einschließlich Ihrer `BulkDeploymentId`.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Rufen Sie nun den Befehl `ListBulkDeploymentDetailedReports` auf, um detaillierte Informationen zu jeder Bereitstellung zu sammeln.

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

Der Befehl sollte einen erfolgreichen Statuscode `200` zurückgeben, zusammen mit einer JSON-Nutzlast aus Informationen:

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

Diese Nutzlast enthält in der Regel eine paginierte Liste jeder Bereitstellung mit ihrem Bereitstellungsstatus, von der neuesten bis zur ältesten. Es enthält auch weitere Informationen für den Fall eines Ausführungsfehlers bei der Sammelbereitstellung. Auch hier sollte die Gesamtzahl der aufgelisteten Bereitstellungen der Anzahl der Gruppen entsprechen, die Sie in Ihrer Eingabedatei für die Sammelbereitstellung angegeben haben.

Die zurückgegebenen Informationen können sich ändern, bis sich die Bereitstellungen in einem Terminalzustand befinden (Erfolg oder Misserfolg). Sie können diesen Befehl bis dahin periodisch aufrufen.

Fehlerbehebung bei Sammelbereitstellungen

Wenn die Sammelbereitstellung nicht erfolgreich ist, können Sie folgende Schritte ausführen, um den Fehler zu beheben. Führen Sie die Befehle in Ihrem Terminal aus.

Fehlerbehebung bei Fehlern in der Eingabedatei

Die Sammelbereitstellung kann bei Syntaxfehlern in der Eingabedatei für die Sammelbereitstellung fehlschlagen. Dies gibt einen Sammelbereitstellungsstatus Failed mit einer Fehlermeldung zurück, die die Zeilennummer des ersten Validierungsfehlers anzeigt. Es gibt vier mögliche Fehlermeldungen:

- InvalidInputFile: Missing *GroupId* at line number: *line number*

Dieser Fehler zeigt an, dass die angegebene Zeile der Eingabedatei den angegebenen Parameter nicht registrieren kann. Die möglichen fehlenden Parameter sind die *GroupId* und die *GroupVersionId*.

- InvalidInputFile: Invalid deployment type at line number : *line number*. Only valid type is 'NewDeployment'.

Dieser Fehler zeigt an, dass in der Zeile der angegebenen Eingabedatei ein ungültiger Bereitstellungstyp angeführt ist. Derzeit wird nur der Bereitstellungstyp *NewDeployment* unterstützt.

- Line *%s* is too long in S3 File. Valid line is less than 256 chars.

Dieser Fehler zeigt an, dass die angegebene Zeile der Eingabedatei zu lang ist und gekürzt werden muss.

- Failed to parse input file at line number: *line number*

Dieser Fehler zeigt an, dass die angegebene Zeile der Eingabedatei als nicht gültig für JSON angesehen wird.

Auf gleichzeitige Sammelbereitstellungen prüfen.

Sie können keine neue Sammelbereitstellung starten, während eine andere noch läuft bzw. sich nicht im terminalen Zustand befindet. Dies kann zum Fehler `Concurrent Deployment Error` führen. Sie können den `ListBulkDeployments`-Befehl verwenden, um zu überprüfen, dass derzeit keine Massenbereitstellung erfolgt. Dieser Befehl listet Ihre Sammelbereitstellungen von der neuesten bis zur ältesten auf.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Verwenden Sie die `BulkDeploymentId` der ersten aufgelisteten Sammelbereitstellung, um den Befehl `GetBulkDeploymentStatus` auszuführen. Wenn sich Ihre jüngste Sammelbereitstellung in einem laufenden Zustand befindet (`Initializing` oder `Running`), verwenden Sie den folgenden Befehl, um die Sammelbereitstellung zu stoppen.

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

Diese Aktion führt zum Status `Stopping`, bis die Bereitstellung auf `Stopped` gesetzt ist. Nachdem die Bereitstellung den Status `Stopped` erreicht hat, können Sie eine neue Sammelbereitstellung starten.

Prüfen ErrorDetails

Führen Sie den Befehl `GetBulkDeploymentStatus` aus, um eine JSON-Nutzlast zurückzugeben, die Informationen über einen Fehler bei der Ausführung der Sammelbereitstellung enthält.

```
"Message": "string",
"ErrorDetails": [
```

```
{
  "DetailedErrorCode": "string",
  "DetailedErrorMessage": "string"
}
```

Bei Beendigung mit einem Fehler, enthalten die `ErrorDetails` der JSON-Nutzlast, die von diesem Aufruf zurückgegeben wird, weitere Informationen über den Fehler bei der Ausführung der Sammelbereitstellung. Ein Fehlerstatuscode in der Serie 400 zeigt beispielsweise einen Eingabefehler an, entweder in den Eingabeparametern oder in den Abhängigkeiten des Anrufers.

Überprüfe das AWS IoT Greengrass Core-Log

Sie können Probleme beheben, indem Sie sich die AWS IoT Greengrass Kernprotokolle ansehen. Verwenden Sie die folgenden Befehle, um das `runtime.log` anzuzeigen:

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

Weitere Informationen zur AWS IoT Greengrass Protokollierung finden Sie unter [Überwachung mit AWS IoT Greengrass Protokollen](#).

Weitere Informationen finden Sie auch unter

Weitere Informationen finden Sie in den folgenden Ressourcen:

- [Stellen Sie AWS IoT Greengrass Gruppen bereit](#)
- [Amazon S3 S3-API-Befehle](#) in der AWS CLI Befehlsreferenz
- [AWS IoT Greengrass Befehle](#) in der AWS CLI Befehlsreferenz

Führen Sie Lambda-Funktionen auf dem Kern aus AWS IoT Greengrass

AWS IoT Greengrass bietet eine containerisierte Lambda-Laufzeitumgebung für benutzerdefinierten Code, in dem Sie schreiben. AWS Lambda Lambda-Funktionen, die für einen AWS IoT Greengrass Kern bereitgestellt werden, werden in der lokalen Lambda-Laufzeit des Kerns ausgeführt. Lokale Lambda-Funktionen können durch lokale Ereignisse, Nachrichten aus der Cloud und andere Quellen ausgelöst werden, wodurch lokale Rechenfunktionen auf Client-Geräten verfügbar sind. Sie können beispielsweise die Funktionen von Greengrass Lambda verwenden, um Gerätedaten zu filtern, bevor Sie die Daten in die Cloud übertragen.

Um eine Lambda-Funktion für einen Core bereitzustellen, fügen Sie die Funktion einer Greengrass-Gruppe hinzu (indem Sie auf die vorhandene Lambda-Funktion verweisen), gruppenspezifische Einstellungen für die Funktion konfigurieren und dann die Gruppe bereitstellen. Wenn die Funktion auf AWS Dienste zugreift, müssen Sie der [Greengrass-Gruppenrolle](#) auch alle erforderlichen Berechtigungen hinzufügen.

Sie können Parameter konfigurieren, die bestimmen, wie die Lambda-Funktionen ausgeführt werden, einschließlich Berechtigungen, Isolierung, Speicherbeschränkungen und mehr. Weitere Informationen finden Sie unter [the section called “Steuerung der Ausführung von Greengrass Lambda-Funktionen”](#).

Note

Diese Einstellungen ermöglichen auch die Ausführung AWS IoT Greengrass in einem Docker-Container. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass In einem Docker-Container ausführen”](#).

In der folgenden Tabelle sind die unterstützten [AWS Lambda Laufzeiten](#) und die Versionen der AWS IoT Greengrass Core-Software aufgeführt, auf denen sie ausgeführt werden können.

Sprache oder Plattform	GGC-Version
Python 3.8	1.11
Python 3.7	1.9 oder höher

Sprache oder Plattform	GGC-Version
Python 2.7 *	1.0 oder später
Java 8	1.1 oder später
Node.js 12.x *	1.10 oder später
Node.js 8.10*	1.9 oder später
Node.js 6.10*	1.1 oder später
C, C++	1.6 oder später

* Sie können Lambda-Funktionen, die diese Laufzeiten verwenden, auf unterstützten Versionen von ausführen AWS IoT Greengrass, aber Sie können sie nicht darin erstellen. AWS Lambda Wenn sich die Laufzeit auf Ihrem Gerät von der für diese Funktion angegebenen AWS Lambda-Laufzeit unterscheidet, können Sie mithilfe von `FunctionRuntimeOverride` in `FunctionDefinitionVersion` Ihre eigene Laufzeit wählen. Weitere Informationen finden Sie unter [CreateFunctionDefinition](#). Weitere Informationen zu unterstützten Laufzeiten finden Sie unter [Runtime-Supportrichtlinie](#) im AWS Lambda Entwicklerhandbuch.

SDKs für Greengrass Lambda-Funktionen

AWS stellt drei bereit SDKs , die von Greengrass Lambda-Funktionen verwendet werden können, die auf einem AWS IoT Greengrass Kern ausgeführt werden. Diese SDKs sind in verschiedenen Paketen enthalten, sodass Funktionen sie gleichzeitig verwenden können. Um ein SDK in einer Greengrass-Lambda-Funktion zu verwenden, nehmen Sie es in das Lambda-Funktions-Bereitstellungspaket auf, in das Sie hochladen. AWS Lambda

AWS IoT Greengrass Kern-SDK

Ermöglicht es lokalen Lambda-Funktionen, mit dem Kern zu interagieren, um:

- Tauschen Sie MQTT-Nachrichten aus mit. AWS IoT Core
- Tauschen Sie MQTT-Nachrichten mit Konnektoren, Client-Geräten und anderen Lambda-Funktionen in der Greengrass-Gruppe aus.
- Interagieren mit dem lokalen Shadow-Service.

- Rufen Sie andere lokale Lambda-Funktionen auf.
- Zugreifen auf [geheime Ressourcen](#).
- Interagieren Sie mit dem [Stream-Manager](#).

AWS IoT Greengrass stellt das AWS IoT Greengrass Core SDK in den folgenden Sprachen und Plattformen zur Verfügung. GitHub

- [AWS IoT Greengrass Kern-SDK SDK for Java](#)
- [AWS IoT Greengrass Kern-SDK für Node.js](#)
- [AWS IoT Greengrass Kern-SDK für Python](#)
- [AWS IoT Greengrass Kern-SDK für C](#)

Um die AWS IoT Greengrass Core-SDK-Abhängigkeit in das Lambda-Funktionsbereitstellungspaket aufzunehmen:

1. Laden Sie die Sprache oder Plattform des AWS IoT Greengrass Core-SDK-Pakets herunter, die der Laufzeit Ihrer Lambda-Funktion entspricht.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der `greengrasssdk`-Ordner.
3. Nehmen Sie es `greengrasssdk` in das Lambda-Funktionsbereitstellungspaket auf, das Ihren Funktionscode enthält. Dies ist das Paket, in das Sie hochladen, AWS Lambda wenn Sie die Lambda-Funktion erstellen.

StreamManagerClient

Nur der folgende AWS IoT Greengrass Core SDKs kann für [Stream Manager-Operationen](#) verwendet werden:

- Java SDK (v1.4.0 oder höher)
- Python SDK (v1.5.0 oder höher)
- Node.js SDK (v1.6.0 oder höher)

Um das AWS IoT Greengrass Core SDK für Python für die Interaktion mit dem Stream Manager zu verwenden, müssen Sie Python 3.7 oder höher installieren. Sie müssen auch Abhängigkeiten installieren, die in Ihre Python-Lambda-Funktionspakete aufgenommen werden sollen:

1. Navigieren Sie zum SDK-Verzeichnis, das die `requirements.txt`-Datei enthält. Diese Datei listet die Abhängigkeiten auf.
2. Installieren Sie die SDK-Abhängigkeiten. Führen Sie beispielsweise den folgenden `pip`-Befehl aus, um sie im aktuellen Verzeichnis zu installieren:

```
pip install --target . -r requirements.txt
```

Installieren Sie das AWS IoT Greengrass Core SDK für Python auf dem Core-Gerät

Wenn Sie Python-Lambda-Funktionen ausführen, können Sie es auch verwenden, [pip](#) das AWS IoT Greengrass Core SDK für Python auf dem Core-Gerät zu installieren. Anschließend können Sie Ihre Funktionen bereitstellen, ohne das SDK in das Lambda-Funktionsbereitstellungspaket aufzunehmen. Weitere Informationen finden Sie unter [greengrasssdk](#).

Diese Unterstützung ist für Cores mit Größenbeschränkungen vorgesehen. Wir empfehlen, dass Sie das SDK nach Möglichkeit in Ihre Lambda-Funktions-Deployment-Pakete aufnehmen.

AWS IoT Greengrass SDK für Machine Learning

Ermöglicht es lokalen Lambda-Funktionen, Modelle für maschinelles Lernen (ML) zu nutzen, die als ML-Ressourcen auf dem Greengrass-Kern bereitgestellt werden. Lambda-Funktionen können das SDK verwenden, um einen lokalen Inferenzdienst aufzurufen und mit ihm zu interagieren, der als Konnektor im Core bereitgestellt wird. Lambda-Funktionen und ML-Konnektoren können das SDK auch verwenden, um Daten zum Hochladen und Veröffentlichen an den ML-Feedback-Konnektor zu senden. Weitere Informationen einschließlich Codebeispielen, die das SDK verwenden, finden Sie unter [the section called “ML-Bildklassifizierung”](#), [the section called “ML-Objekterkennung”](#) und [the section called “ML-Feedback”](#).

In der folgenden Tabelle sind die unterstützten Sprachen oder Plattformen für SDK-Versionen sowie die Versionen der AWS IoT Greengrass Core-Software aufgeführt, auf denen sie ausgeführt werden können.

SDK-Version	Sprache oder Plattform	Erforderliche GGC-Version	Änderungsprotokoll
1.1.0	Python 3.7 oder 2.7	1.9.3 oder höher	Unterstützung für Python 3.7 und neuer feedback-Client hinzugefügt.
1.0.0	Python 2.7	1.7 oder später	Erstversion.

Informationen zum Download finden Sie unter [the section called “AWS IoT Greengrass ML SDK-Software”](#).

AWS SDKs

Ermöglicht lokalen Lambda-Funktionen direkte Aufrufe von AWS Diensten wie Amazon S3, DynamoDB und AWS IoT. AWS IoT Greengrass Um ein AWS SDK in einer Greengrass Lambda-Funktion zu verwenden, müssen Sie es in Ihr Bereitstellungspaket aufnehmen. Wenn Sie das AWS SDK im selben Paket wie das AWS IoT Greengrass Core-SDK verwenden, stellen Sie sicher, dass Ihre Lambda-Funktionen die richtigen Namespaces verwenden. Die Funktionen von Greengrass Lambda können nicht mit Cloud-Diensten kommunizieren, wenn der Kern offline ist.

Laden Sie das AWS SDKs aus dem [Getting Started Resource Center](#) herunter.

Weitere Informationen zum Erstellen eines Bereitstellungspakets finden Sie [the section called “Eine Lambda-Funktion erstellen und verpacken”](#) im Tutorial Erste Schritte oder unter [Erstellen eines Bereitstellungspakets](#) im AWS Lambda Entwicklerhandbuch.

Migration von Cloud-basierten Lambda-Funktionen

Das AWS IoT Greengrass Core SDK folgt dem AWS SDK-Programmiermodell, das es einfach macht, Lambda-Funktionen, die für die Cloud entwickelt wurden, auf Lambda-Funktionen zu portieren, die auf einem AWS IoT Greengrass Core ausgeführt werden.

Die folgende Python-Lambda-Funktion verwendet beispielsweise die AWS SDK für Python (Boto3), um eine Nachricht zum Thema `some/topic` in der Cloud zu veröffentlichen:

```
import boto3

iot_client = boto3.client("iot-data")
response = iot_client.publish(
    topic="some/topic", qos=0, payload="Some payload".encode()
)
```

Um die Funktion für einen AWS IoT Greengrass Kern zu portieren, ändern Sie in der `import` Anweisung und der `client` Initialisierung den `boto3` Modulnamen in `greengrasssdk`, wie im folgenden Beispiel gezeigt:

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

Note

Das AWS IoT Greengrass Core SDK unterstützt nur das Senden von MQTT-Nachrichten mit QoS = 0. Weitere Informationen finden Sie unter [the section called “Nachrichtenqualität des Service”](#).

Die Ähnlichkeit zwischen den Programmiermodellen ermöglicht es Ihnen auch, Ihre Lambda-Funktionen in der Cloud zu entwickeln und sie dann AWS IoT Greengrass mit minimalem Aufwand dorthin zu migrieren. [Ausführbare Lambda-Dateien](#) werden nicht in der Cloud ausgeführt, sodass Sie das AWS SDK nicht verwenden können, um sie vor der Bereitstellung in der Cloud zu entwickeln.

Lambda-Funktionen nach Alias oder Version referenzieren

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion. Aliase werden bei der Gruppenbereitstellung zu Versionsnummern aufgelöst. Bei Verwendung eines Alias wird die aufgelöste Version auf die Version aktualisiert, auf die das Alias zum Zeitpunkt der Bereitstellung weist.

AWS IoT Greengrass unterstützt keine Lambda-Aliase für `$LATEST`-Versionen. `$LATEST`-Versionen sind nicht an unveränderliche, veröffentlichte Funktionsversionen gebunden und können jederzeit geändert werden, was dem Prinzip der AWS IoT Greengrass Versionsunveränderlichkeit widerspricht.

Eine gängige Praxis, um Ihre Greengrass Lambda-Funktionen bei Codeänderungen auf dem neuesten Stand zu halten, besteht darin, einen Alias zu verwenden, der **PRODUCTION** in Ihrer Greengrass-Gruppe und Ihren Abonnements benannt ist. Wenn Sie neue Versionen Ihrer Lambda-Funktion in die Produktion hochstufen, verweisen Sie den Alias auf die neueste stabile Version und stellen Sie die Gruppe dann erneut bereit. Sie können diese Methode auch für ein Rollback auf eine frühere Version verwenden.

Steuerung der Ausführung von Greengrass Lambda-Funktionen mithilfe einer gruppenspezifischen Konfiguration

AWS IoT Greengrass bietet Cloud-basiertes Management von Greengrass Lambda-Funktionen. Obwohl der Code und die Abhängigkeiten einer Lambda-Funktion mithilfe verwaltet werden, können Sie konfigurieren AWS Lambda, wie sich die Lambda-Funktion verhält, wenn sie in einer Greengrass-Gruppe ausgeführt wird.

Gruppenspezifische Konfigurationseinstellungen

AWS IoT Greengrass bietet die folgenden gruppenspezifischen Konfigurationseinstellungen für Greengrass Lambda-Funktionen.

Systembenutzer und Gruppe

Die Zugriffsidentität, die zum Ausführen einer Lambda-Funktion verwendet wird. Standardmäßig werden Lambda-Funktionen als [Standardzugriffsidentität](#) der Gruppe ausgeführt. In der Regel ist dies das standardmäßige AWS IoT Greengrass -Systemkonto (ggc_user und ggc_group). Sie können die Einstellung ändern und die Benutzer-ID und Gruppen-ID auswählen, die über die für die Ausführung der Lambda-Funktion erforderlichen Berechtigungen verfügen. Sie können sowohl die UID als auch die GID oder durch Leerlassen eines der Felder nur eine davon überschreiben. Diese Einstellung bietet Ihnen eine genauere Kontrolle über den Zugriff auf Geräte-Ressourcen. Wir empfehlen Ihnen, Ihre Greengrass-Hardware mit entsprechenden Ressourcenlimits, Dateiberechtigungen und Festplattenkontingenten für die Benutzer und Gruppen zu konfigurieren, deren Berechtigungen zur Ausführung von Lambda-Funktionen verwendet werden.

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Important

Wir empfehlen, Lambda-Funktionen nicht als Root auszuführen, sofern dies nicht unbedingt erforderlich ist. Die Ausführung als Root erhöht die folgenden Risiken:

- Das Risiko unbeabsichtigter Änderungen, wie z. B. das versehentliche Löschen einer wichtigen Datei.
- Das Risiko für Ihre Daten und Ihr Gerät durch böswillige Personen.
- Das Risiko von Containern entgeht, wenn Docker-Container mit `--net=host` und ausgeführt werden. `UID=EUID=0`

Wenn Sie die Ausführung als Root ausführen müssen, müssen Sie die AWS IoT Greengrass Konfiguration aktualisieren, um sie zu aktivieren. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Systembenutzer-ID (Nummer)

Die Benutzer-ID für den Benutzer, der über die erforderlichen Berechtigungen verfügt, um die Lambda-Funktion auszuführen. Diese Einstellung ist nur verfügbar, wenn Sie die Ausführung mit einer anderen Benutzer-ID/Gruppen-ID auswählen. Sie können den `getent passwd` Befehl auf Ihrem AWS IoT Greengrass Core-Gerät verwenden, um die Benutzer-ID nachzuschlagen, die Sie zum Ausführen der Lambda-Funktion verwenden möchten.

Wenn Sie dieselbe UID verwenden, um Prozesse und die Lambda-Funktion auf einem Greengrass-Core-Gerät auszuführen, kann Ihre Greengrass-Gruppenrolle den Prozessen temporäre Anmeldeinformationen gewähren. Die Prozesse können die temporären Anmeldeinformationen in allen Greengrass-Kernbereitstellungen verwenden.

Systemgruppen-ID (Nummer)

Die Gruppen-ID für die Gruppe, die über die erforderlichen Berechtigungen verfügt, um die Lambda-Funktion auszuführen. Diese Einstellung ist nur verfügbar, wenn Sie die Ausführung mit einer anderen ID/group Benutzer-ID auswählen. Sie können den `getent group` Befehl auf Ihrem AWS IoT Greengrass Core-Gerät verwenden, um die Gruppen-ID nachzuschlagen, die Sie zum Ausführen der Lambda-Funktion verwenden möchten.

Containerisierung der Lambda-Funktion

Wählen Sie aus, ob die Lambda-Funktion mit der Standardcontainerisierung für die Gruppe ausgeführt wird, oder geben Sie die Containerisierung an, die immer für diese Lambda-Funktion verwendet werden soll.

Der Containerisierungsmodus einer Lambda-Funktion bestimmt ihren Isolationsgrad.

- Containerisierte Lambda-Funktionen werden im Greengrass-Container-Modus ausgeführt. Die Lambda-Funktion wird in einer isolierten Laufzeitumgebung (oder einem Namespace) innerhalb des AWS IoT Greengrass Containers ausgeführt.
- Nicht containerisierte Lambda-Funktionen werden im Modus `Kein Container` ausgeführt. Die Lambda-Funktionen laufen als regulärer Linux-Prozess ohne jegliche Isolierung.

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Wir empfehlen, dass Sie Lambda-Funktionen in einem Greengrass-Container ausführen, es sei denn, Ihr Anwendungsfall erfordert, dass sie ohne Containerisierung ausgeführt werden. Wenn Ihre Lambda-Funktionen in einem Greengrass-Container ausgeführt werden, können Sie angehängte lokale Ressourcen und Geräteressourcen nutzen und von den Vorteilen der Isolierung und erhöhten Sicherheit profitieren. Bevor Sie die Containerisierung ändern, schlagen Sie unter [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#) nach.

Note

Damit Sie ohne Aktivierung Ihres Gerätekernel-Namespace und Ihrer Cgroup ausgeführt werden können, müssen alle Ihre Lambda-Funktionen ohne Containerisierung

ausgeführt werden. Sie können dies leicht erreichen, indem Sie für die Gruppe die Standardeinstellung der Containerisierung festlegen. Weitere Informationen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).

Speicherlimit

Die Speicherzuweisung für die Funktion. Der Standardwert ist 16 MB.

Note

Die Einstellung für das Speicherlimit ist nicht mehr verfügbar, wenn Sie die Lambda-Funktion so ändern, dass sie ohne Containerisierung ausgeführt wird. Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, haben kein Speicherlimit. Die Einstellung für das Speicherlimit wird verworfen, wenn Sie die Standardeinstellung für die Containerisierung der Lambda-Funktion oder Gruppe so ändern, dass sie ohne Containerisierung ausgeführt wird.

Timeout (Zeitüberschreitung)

Die Dauer vor Beendigung der Funktion oder Anforderung. Der Standardwert ist 3 Sekunden.

Angeheftet

Ein Lambda-Funktionslebenszyklus kann bedarfsgerecht oder langlebig sein. Die Standardeinstellung ist „On-Demand“.

Eine On-Demand-Lambda-Funktion startet in einem neuen oder wiederverwendeten Container, wenn sie aufgerufen wird. Anforderungen an die Funktion werden möglicherweise von allen verfügbaren Containern verarbeitet. Eine langlebige — oder fixierte — Lambda-Funktion wird nach dem Start automatisch AWS IoT Greengrass gestartet und läuft weiterhin in ihrem eigenen Container (oder Sandbox). Alle Anforderungen an die Funktion werden vom gleichen Container verarbeitet. Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

Lesezugriff auf das /sys-Verzeichnis

Ob die Funktion auf den /sys-Ordner des Hosts zugreifen kann. Verwenden Sie diese Einstellung, wenn die Funktion Geräteinformationen aus /sys lesen muss. Der Standardwert lautet „false“.

 Note

Diese Einstellung ist nicht verfügbar, wenn Sie eine Lambda-Funktion ohne Containerisierung ausführen. Der Wert dieser Einstellung wird verworfen, wenn Sie die Lambda-Funktion so ändern, dass sie ohne Containerisierung ausgeführt wird.

Art der Kodierung

Der erwartete Kodierungstyp der Eingabenutzlast für die Funktion: JSON oder Binär. Der Standardwert ist JSON.

Support für den binären Kodierungstyp ist ab AWS IoT Greengrass Core Software v1.5.0 und AWS IoT Greengrass Core SDK v1.1.0 verfügbar. Bei Funktionen die bei der Gerätedaten interagieren, kann es nützlich sein, binäre Eingabedaten zu akzeptieren, da es die eingeschränkten Hardware-Funktionen von Geräten oft schwierig oder gar unmöglich machen, einen JSON-Datentyp zu erstellen.

 Note

[Ausführbare Lambda-Dateien](#) unterstützen nur den binären Kodierungstyp, nicht JSON.

Prozessargumente

Die Befehlszeilenargumente werden an die Lambda-Funktion übergeben, wenn sie ausgeführt wird.

Umgebungsvariablen

Schlüssel-Wert-Paare, die Einstellungen dynamisch an Funktionscode und -bibliotheken übergeben. Lokale Umgebungsvariablen funktionieren auf die gleiche Weise wie [AWS Lambda - Funktionsumgebungsvariablen](#), sind aber in der Core-Umgebung verfügbar.

Resource access policies (Richtlinien für Ressourcenzugriff)

Eine Liste mit bis zu 10 [lokalen Ressourcen](#), [geheimen Ressourcen und Ressourcen](#) für [maschinelles Lernen](#), auf die die Lambda-Funktion zugreifen darf, sowie der entsprechenden read-only read-write OR-Berechtigung. In der Konsole werden diese verbundenen Ressourcen auf der Gruppenkonfigurationsseite auf der Registerkarte Ressourcen aufgeführt.

Der [Containerisierungsmodus](#) beeinflusst, wie Lambda-Funktionen auf lokale Geräte- und Volume-Ressourcen sowie auf Ressourcen für maschinelles Lernen zugreifen können.

- Lambda-Funktionen, die nicht containerisiert sind, müssen direkt über das Dateisystem auf dem Kerngerät auf lokale Geräte- und Volume-Ressourcen zugreifen.
- Um nicht containerisierten Lambda-Funktionen den Zugriff auf maschinelle Lernressourcen in der Greengrass-Gruppe zu ermöglichen, müssen Sie die Eigenschaften Ressourcenbesitzer und Zugriffsberechtigungen für die maschinelle Lernressource festlegen. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Machine Learning-Ressourcen”](#).

Informationen zur Verwendung der AWS IoT Greengrass API zum Festlegen gruppenspezifischer Konfigurationseinstellungen für benutzerdefinierte Lambda-Funktionen finden Sie [CreateFunctionDefinition](#) in der AWS IoT Greengrass Version 1 API-Referenz oder [create-function-definition](#) in der Befehlsreferenz. AWS CLI Um Lambda-Funktionen auf einem Greengrass-Kern bereitzustellen, erstellen Sie eine Funktionsdefinitionsversion, die Ihre Funktionen enthält, erstellen Sie eine Gruppenversion, die auf die Funktionsdefinitionsversion und andere Gruppenkomponenten verweist, und [stellen Sie dann die](#) Gruppe bereit.

Eine Lambda-Funktion als Root ausführen

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Bevor Sie eine oder mehrere Lambda-Funktionen als Root ausführen können, müssen Sie zuerst die AWS IoT Greengrass Konfiguration aktualisieren, um die Unterstützung zu aktivieren. Die Support für die Ausführung von Lambda-Funktionen als Root ist standardmäßig deaktiviert. Die Bereitstellung schlägt fehl, wenn Sie versuchen, eine Lambda-Funktion bereitzustellen und sie als Root (UID und GID von 0) auszuführen, und Sie die Konfiguration nicht aktualisiert haben. AWS IoT Greengrass Im Laufzeitprotokoll (*greengrass_root*/ggc/var/log/system/runtime.log) wird ein Fehler wie der folgende angezeigt:

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
configured to run lambdas with root permissions
```

Important

Wir empfehlen, Lambda-Funktionen nicht als Root auszuführen, sofern dies nicht unbedingt erforderlich ist. Die Ausführung als Root erhöht die folgenden Risiken:

- Das Risiko unbeabsichtigter Änderungen, wie z. B. das versehentliche Löschen einer wichtigen Datei.
- Das Risiko für Ihre Daten und Ihr Gerät durch böswillige Personen.
- Das Risiko von Containern entgeht, wenn Docker-Container mit `--net=host` und ausgeführt werden. `UID=EUID=0`

Um zu ermöglichen, dass Lambda-Funktionen als Root ausgeführt werden

1. Navigieren Sie auf Ihrem AWS IoT Greengrass Gerät zum Ordner *greengrass-root* /config.

 Note

Standardmäßig *greengrass-root* ist dies das Verzeichnis /greengrass.

2. Bearbeiten Sie die Datei config.json, indem Sie "allowFunctionsToRunAsRoot" : "yes" zum Feld runtime hinzufügen. Beispiel:

```
{
  "coreThing" : {
    ...
  },
  "runtime" : {
    ...
    "allowFunctionsToRunAsRoot" : "yes"
  },
  ...
}
```

3. Verwenden Sie die folgenden Befehle, um neu zu starten: AWS IoT Greengrass

```
cd /greengrass/ggc/core
sudo ./greengrassd restart
```

Jetzt können Sie die Benutzer-ID und Gruppen-ID (UID/GID) von Lambda-Funktionen auf 0 setzen, um diese Lambda-Funktion als Root auszuführen.

Sie können den Wert von `"allowFunctionsToRunAsRoot"` zu ändern `"no"` und neu starten, AWS IoT Greengrass wenn Sie verhindern möchten, dass Lambda-Funktionen als Root ausgeführt werden.

Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Standardmäßig werden Lambda-Funktionen in einem AWS IoT Greengrass Container ausgeführt. Dieser Container schafft eine Isolierung zwischen Ihren Funktionen und dem Host und bietet zusätzliche Sicherheit sowohl für den Host als auch für die Funktionen im Container.

Wir empfehlen, dass Sie Lambda-Funktionen in einem Greengrass-Container ausführen, es sei denn, Ihr Anwendungsfall erfordert, dass sie ohne Containerisierung ausgeführt werden. Wenn Sie Ihre Lambda-Funktionen in einem Greengrass-Container ausführen, haben Sie mehr Kontrolle über die Beschränkung des Zugriffs auf Ressourcen.

Es folgen einige Beispiel-Anwendungsfälle für die Ausführung ohne Containerisierung:

- Sie möchten AWS IoT Greengrass auf einem Gerät laufen, das den Container-Modus nicht unterstützt (z. B. weil Sie eine spezielle Linux-Distribution verwenden oder eine zu alte Kernelversion haben).
- Sie möchten Ihre Lambda-Funktion in einer anderen Containerumgebung mit eigenem OverlayFS ausführen, stoßen aber bei der Ausführung in einem Greengrass-Container auf OverlayFS-Konflikte.
- Sie benötigen Zugriff auf lokale Ressourcen, deren Pfade zum Zeitpunkt der Bereitstellung nicht bestimmt werden oder sich nach der Bereitstellung ändern können, wie z. B. Plug-In-Geräte.
- Sie haben eine Legacy-Anwendung, die als Prozess geschrieben wurde, und bei der Ausführung als containerisierte Lambda-Funktion sind Probleme aufgetreten.

Containerisierungsunterschiede

Containerisierung	Hinweise
Greengrass-Container	• Alle AWS IoT Greengrass Funktionen sind verfügbar, wenn Sie eine Lambda-Funktion in einem Greengrass-Container ausführen.

Containerisierung	Hinweise
	• Lambda-Funktionen, die in einem Greengrass-Container ausgeführt werden, haben keinen Zugriff auf den bereitgestellten Code anderer Lambda-Funktionen, selbst wenn sie mit derselben Gruppen-ID ausgeführt werden. Mit anderen Worten, Ihre Lambda-Funktionen laufen stärker voneinander isoliert. • Da bei Lambda-Funktionen, die in einem AWS IoT Greengrass Container ausgeführt werden, alle untergeordneten Prozesse im selben Container wie die Lambda-Funktion ausgeführt werden, werden die untergeordneten Prozesse beendet, wenn die Lambda-Funktion beendet wird.

Containerisierung	Hinweise
Kein Container	• Die folgenden Funktionen sind für Lambda-Funktionen, die nicht containerisiert sind, nicht verfügbar: • Speichergrenzen für Lambda-Funktionen. • Lokale Geräte- und Volume-Ressourcen Sie müssen direkt auf diese Ressourcen auf dem Core-Gerät zugreifen, anstatt als Mitglieder der Greengrass-Gruppe auf sie zuzugreifen. • Wenn Ihre nicht containerisierte Lambda-Funktion auf eine maschinelle Lernressource zugreift, müssen Sie einen Ressourcenbesitzer identifizieren und Zugriffsberechtigungen für die Ressource festlegen, nicht für die Lambda-Funktion. Dies erfordert Core-Software v1.10 oder höher AWS IoT Greengrass . Weitere Informationen finden Sie unter the section called “Zugreifen auf Machine Learning-Ressourcen”. • Die Lambda-Funktion hat schreibgeschützten Zugriff auf den bereitgestellten Code anderer Lambda-Funktionen, die mit derselben Gruppen-ID ausgeführt werden. • Lambda-Funktionen, die untergeordnete Prozesse in einer anderen Prozesssitzung oder mit einem überschriebenen SIGHUP-Handler (Signal Hangup) erzeugen, z. B. mit dem Hilfsprogramm nohup, werden nicht automatisch beendet, wenn die übergeordnete Lambda-Funktion beendet wird. AWS IoT Greengrass

 Note

Die Standardeinstellung für Containerisierung für die Greengrass-Gruppe gilt nicht für [Konnektoren](#).

Das Ändern der Containerisierung für eine Lambda-Funktion kann zu Problemen bei der Bereitstellung führen. Wenn Sie Ihrer Lambda-Funktion lokale Ressourcen zugewiesen haben, die mit Ihren neuen Containerisierungseinstellungen nicht mehr verfügbar sind, schlägt die Bereitstellung fehl.

- Wenn Sie ändern, dass eine Lambda-Funktion nicht mehr in einem Greengrass-Container ausgeführt wird, sondern ohne Containerisierung ausgeführt wird, werden die Speicherlimits für die Funktion verworfen. Sie müssen direkt auf das Dateisystem zugreifen, anstatt verknüpfte lokale Ressourcen zu verwenden. Vor der Bereitstellung müssen Sie alle verknüpften Ressourcen entfernen.
- Wenn Sie eine Lambda-Funktion von der Ausführung ohne Containerisierung auf die Ausführung in einem Container ändern, verliert Ihre Lambda-Funktion den direkten Zugriff auf das Dateisystem. Sie müssen für jede Funktion ein Speicherlimit definieren oder den Standard 16 MB übernehmen. Sie können diese Einstellungen für jede Lambda-Funktion vor der Bereitstellung konfigurieren.

So ändern Sie die Containerisierungseinstellungen für eine Lambda-Funktion

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe aus, die die Lambda-Funktion enthält, deren Einstellungen Sie ändern möchten.
3. Wählen Sie die Registerkarte Lambda-Funktionen.
4. Wählen Sie in der Lambda-Funktion, die Sie ändern möchten, die Ellipse (...) und dann Konfiguration bearbeiten aus.
5. Ändern Sie die Containerisierungseinstellungen. Wenn Sie die Lambda-Funktion für die Ausführung in einem Greengrass-Container konfigurieren, müssen Sie auch das Speicherlimit und den Lesezugriff auf das Verzeichnis /sys festlegen.
6. Wählen Sie Speichern und dann Bestätigen, um die Änderungen an Ihrer Lambda-Funktion zu speichern.

Die Änderungen werden wirksam, wenn die Gruppe bereitgestellt wird.

Sie können auch das [CreateFunctionDefinition](#) und [CreateFunctionDefinitionVersion](#) in der AWS IoT Greengrass API-Referenz verwenden. Wenn Sie die Containerisierungseinstellung ändern, müssen Sie auch die anderen Parameter aktualisieren. Wenn Sie beispielsweise von der Ausführung einer Lambda-Funktion in einem Greengrass-Container zur Ausführung ohne Containerisierung wechseln, müssen Sie den Parameter löschen. `MemorySize`

Bestimmen der von Ihrem Greengrass-Gerät unterstützten Isolierungsmodi

Sie können die AWS IoT Greengrass Abhängigkeitsprüfung verwenden, um festzustellen, welche Isolationsmodi (`container/no Greengrass-Container`) von Ihrem Greengrass-Gerät unterstützt werden.

Um die Abhängigkeitsprüfung auszuführen AWS IoT Greengrass

1. [Laden Sie den AWS IoT Greengrass Abhängigkeitsprüfer aus dem Repository herunter und führen Sie ihn aus. GitHub](#)

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

2. Wenn `more` erscheint, drücken Sie die Spacebar-Taste, um eine weitere Textseite anzuzeigen.

Weitere Information zum Befehl `modprobe` erhalten Sie, wenn Sie `man modprobe` im Terminal ausführen.

Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe

Diese Funktion ist für AWS IoT Greengrass Core v1.8 und höher verfügbar.

Für mehr Kontrolle über den Zugriff auf Geräteressourcen können Sie die Standardzugriffsidentität konfigurieren, die für die Ausführung von Lambda-Funktionen in der Gruppe verwendet wird. Diese Einstellung bestimmt die Standardberechtigungen, die Ihren Lambda-Funktionen erteilt werden, wenn sie auf dem Kerngerät ausgeführt werden. Um die Einstellung für einzelne Funktionen in der Gruppe

zu überschreiben, können Sie die Run as (Ausführen als)-Eigenschaft der Gruppe verwenden. Weitere Informationen hierzu finden Sie unter [Run as \(Ausführen als\)](#).

Diese Einstellung auf Gruppenebene wird auch für die Ausführung der zugrunde liegenden AWS IoT Greengrass Core-Software verwendet. Dies besteht aus Lambda-Systemfunktionen, die Vorgänge wie Nachrichtenweiterleitung, lokale Schattensynchronisierung und automatische IP-Adresserkennung verwalten.

Die Standardzugriffsidentität kann so konfiguriert werden, dass sie als AWS IoT Greengrass Standardsystemkonten (ggc_user und ggc_group) ausgeführt wird oder die Berechtigungen eines anderen Benutzers oder einer anderen Gruppe verwendet. Wir empfehlen Ihnen, Ihre Greengrass-Hardware mit entsprechenden Ressourcenlimits, Dateiberechtigungen und Festplattenkontingenten für alle Benutzer und Gruppen zu konfigurieren, deren Berechtigungen zur Ausführung von benutzerdefinierten Lambda-Funktionen oder System-Lambda-Funktionen verwendet werden.

Um die Standardzugriffsidentität für Ihre Gruppe zu ändern AWS IoT Greengrass

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe aus, deren Einstellungen Sie ändern möchten.
3. Wählen Sie die Registerkarte Lambda-Funktionen und wählen Sie im Abschnitt Standardlaufzeitumgebung für Lambda-Funktionen die Option Bearbeiten aus.
4. Wählen Sie auf der Seite Standard-Laufzeitumgebung für Lambda-Funktionen bearbeiten unter Standardsystembenutzer und -gruppe die Option Andere Benutzer-ID/Gruppen-ID aus.

Wenn Sie diese Option wählen, werden die Felder Systembenutzer-ID (Nummer) und Systemgruppen-ID (Nummer) angezeigt.

5. Geben Sie eine Benutzer-ID, eine Gruppen-ID oder beides ein. Wenn Sie ein Feld leer lassen, wird das entsprechende Greengrass-Systemkonto (ggc_user oder ggc_group) verwendet.
 - Geben Sie unter Systembenutzer-ID (Nummer) die Benutzer-ID für den Benutzer ein, der über die Berechtigungen verfügt, die Sie standardmäßig zum Ausführen von Lambda-Funktionen in der Gruppe verwenden möchten. Mit dem Befehl `getent passwd` auf Ihrem AWS IoT Greengrass -Gerät können Sie die Benutzer-ID abrufen.
 - Geben Sie unter Systemgruppen-ID (Nummer) die Gruppen-ID für die Gruppe ein, die über die Berechtigungen verfügt, die Sie standardmäßig zum Ausführen von Lambda-Funktionen in der Gruppe verwenden möchten. Mit dem Befehl `getent group` auf Ihrem AWS IoT Greengrass -Gerät können Sie die Gruppen-ID abrufen.

 Important

Die Ausführung als Root-Benutzer erhöht die Risiken für Ihre Daten und Geräte. Nicht als Root-Benutzer (UID/GID = 0) ausführen, sofern es Ihr Business Case nicht erfordert. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Die Änderungen werden wirksam, wenn die Gruppe bereitgestellt wird.

Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Die Containerisierungseinstellung für eine Greengrass-Gruppe bestimmt die Standardcontainerisierung für die Lambda-Funktionen in der Gruppe.

- Im Greengrass-Container-Modus werden Lambda-Funktionen standardmäßig in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt.
- Im Modus Kein Container werden Lambda-Funktionen standardmäßig als reguläre Linux-Prozesse ausgeführt.

Sie können Gruppeneinstellungen ändern, um die Standardcontainerisierung für Lambda-Funktionen in der Gruppe anzugeben. Sie können diese Einstellung für eine oder mehrere Lambda-Funktionen in der Gruppe überschreiben, wenn Sie möchten, dass die Lambda-Funktionen mit einer anderen Containerisierung als der Gruppenstandardeinstellung ausgeführt werden. Bevor Sie die Containerisierungseinstellungen ändern, schlagen Sie unter [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#) nach.

 Important

Wenn Sie die Standardcontainerisierung für die Gruppe ändern möchten, aber über eine oder mehrere Funktionen verfügen, die eine andere Containerisierung verwenden, ändern Sie die Einstellungen für die Lambda-Funktionen, bevor Sie die Gruppeneinstellung ändern. Wenn

Sie zuerst die Containerisierungseinstellung der Gruppe ändern, werden die Werte für die Einstellungen Speicherlimit und Lesezugriff auf das /sys-Verzeichnis verworfen.

Um die Containerisierungseinstellungen für Ihre Gruppe zu ändern AWS IoT Greengrass

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe aus, deren Einstellungen Sie ändern möchten.
3. Wählen Sie die Registerkarte Lambda-Funktionen.
4. Wählen Sie unter Standard-Laufzeitumgebung für Lambda-Funktionen die Option Bearbeiten aus.
5. Ändern Sie auf der Seite Standard-Lambda-Funktions-Laufzeitumgebung bearbeiten unter Standardcontainerisierung von Lambda-Funktionen die Einstellung für die Containerisierung.
6. Wählen Sie Speichern.

Die Änderungen werden wirksam, wenn die Gruppe bereitgestellt wird.

Kommunikationsflüsse für Greengrass Lambda-Funktionen

Die Funktionen von Greengrass Lambda unterstützen verschiedene Methoden der Kommunikation mit anderen Mitgliedern der AWS IoT Greengrass Gruppe, lokalen Diensten und Cloud-Diensten (einschließlich AWS Diensten).

Kommunikation über MQTT-Nachrichten

Lambda-Funktionen können MQTT-Nachrichten mithilfe eines Publish-Subscribe-Musters senden und empfangen, das durch Abonnements gesteuert wird.

Dieser Kommunikationsfluss ermöglicht es Lambda-Funktionen, Nachrichten mit den folgenden Entitäten auszutauschen:

- Client-Geräte in der Gruppe.
- Konnektoren in der Gruppe.
- Andere Lambda-Funktionen in der Gruppe.

- AWS IoT.
- Lokaler Geräteschattenservice.

Ein Abonnement definiert eine Nachrichtenquelle, ein Meldeziel und ein Thema (oder Objekt), das verwendet wird, um Nachrichten von der Quelle zum Ziel weiterzuleiten. Nachrichten, die in einer Lambda-Funktion veröffentlicht werden, werden an den registrierten Handler der Funktion übergeben. Abonnements bieten mehr Sicherheit und vorhersehbare Interaktionen. Weitere Informationen finden Sie unter [the section called “Verwaltete Abonnements im MQTT Messaging-Workflow”](#).

Note

Wenn der Core offline ist, können die Funktionen von Greengrass Lambda Nachrichten mit Client-Geräten, Konnektoren, anderen Funktionen und lokalen Shadows austauschen, aber Nachrichten an AWS IoT werden in die Warteschlange gestellt. Weitere Informationen finden Sie unter [the section called “MQTT-Nachrichtenwarteschlange”](#).

Andere Kommunikationsströme

- Um mit lokalen Geräte- und Volumenressourcen sowie Modellen für maschinelles Lernen auf einem Kerngerät zu interagieren, verwenden die Funktionen von Greengrass Lambda plattformspezifische Betriebssystemschnittstellen. Beispielsweise können Sie die `open`-Methode im `os`-Modul in Python-Funktionen verwenden. Damit eine Funktion auf eine Ressource zugreifen kann, muss die Funktion der Ressource zugeordnet und die Berechtigungen `read-only` oder `read-write` müssen gewährt werden. Weitere Informationen, einschließlich der Verfügbarkeit von AWS IoT Greengrass Kernversionen, finden Sie unter und. [Zugreifen auf lokale Ressourcen the section called “Zugreifen auf Ressourcen für maschinelles Lernen über den Lambda-Funktionscode”](#)

Note

Wenn Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, können Sie keine angeschlossenen lokalen Geräte- und Volume-Ressourcen verwenden und müssen direkt auf diese Ressourcen zugreifen.

- Lambda-Funktionen können den Lambda Client im AWS IoT Greengrass Core SDK verwenden, um andere Lambda-Funktionen in der Greengrass-Gruppe aufzurufen.

- Lambda-Funktionen können das AWS SDK verwenden, um mit AWS Diensten zu kommunizieren. Weitere Informationen finden Sie unter [AWS SDK](#).
- Lambda-Funktionen können Schnittstellen von Drittanbietern verwenden, um mit externen Cloud-Diensten zu kommunizieren, ähnlich wie cloudbasierte Lambda-Funktionen.

Note

Die Funktionen von Greengrass Lambda können nicht mit AWS oder anderen Cloud-Diensten kommunizieren, wenn der Kern offline ist.

Abrufen des Eingabe-MQTT-Themas (oder -Betreffs)

AWS IoT Greengrass verwendet Abonnements, um den Austausch von MQTT-Nachrichten zwischen Client-Geräten, Lambda-Funktionen und Konnektoren in einer Gruppe und mit AWS IoT oder dem lokalen Shadow-Service zu steuern. Abonnements definieren eine Nachrichtenquelle, ein Nachrichtenziel und ein MQTT-Thema, die zum Routen von Nachrichten verwendet werden. Wenn das Ziel eine Lambda-Funktion ist, wird der Handler der Funktion aufgerufen, wenn die Quelle eine Nachricht veröffentlicht. Weitere Informationen finden Sie unter [the section called “Kommunikation über MQTT-Nachrichten”](#).

Das folgende Beispiel zeigt, wie eine Lambda-Funktion das Eingabethema aus dem abrufen kann `context`, das an den Handler übergeben wurde. Dies geschieht durch Zugriff auf den `subject`-Schlüssel aus der Kontexthierarchie (`context.client_context.custom['subject']`). In diesem Beispiel wird außerdem die Eingabe-JSON-Nachricht geparkt und anschließend das geparkte Thema und die Nachricht veröffentlicht.

Note

In der AWS IoT Greengrass API wird das Thema eines [Abonnements](#) durch die `subject` Eigenschaft repräsentiert.

```
import greengrasssdk
import logging
```

```

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

OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
input_message)
        logging.info(response)
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return

```

Um die Funktion zu testen, fügen Sie sie mithilfe der Standardkonfigurationseinstellungen Ihrer Gruppe hinzu. Fügen Sie anschließend die folgenden Abonnements hinzu und stellen Sie die Gruppe bereit. Detaillierte Anweisungen finden Sie unter [the section called “Modul 3 \(Teil 1\): Lambda-Funktionen auf AWS IoT Greengrass”](#).

Target
ter

Event
Cloud


```
test/
input_message

test/
topic_results
```

Nach Abschluss der Bereitstellung rufen Sie die Funktion auf.

1. Öffnen Sie in der AWS IoT Konsole die MQTT-Testclient-Seite.
2. Abonnieren Sie das `test/topic_results` Thema, indem Sie die Registerkarte Thema abonnieren auswählen.
3. Veröffentlichen Sie eine Nachricht zu dem `test/input_message` Thema, indem Sie die Registerkarte In einem Thema veröffentlichen auswählen. In diesem Beispiel müssen Sie die `test-key`-Eigenschaft in die JSON-Nachricht einbinden.

```
{
  "test-key": "Some string value"
}
```

Ist der Befehl erfolgreich, veröffentlicht die Funktion die Eingabethema- und Nachrichtenzeichenfolge im `test/topic_results`-Thema.

Lebenszykluskonfiguration für Greengrass Lambda-Funktionen

Der Greengrass Lambda-Funktionslebenszyklus bestimmt, wann eine Funktion gestartet wird und wie sie Container erstellt und verwendet. Der Lebenszyklus legt außerdem fest, wie Variablen und die Vorverarbeitungslogik außerhalb des Funktionshandlers beibehalten werden.

AWS IoT Greengrass unterstützt On-Demand-Lebenszyklen (Standard) oder langlebige Lebenszyklen:

- On-Demand-Funktionen starten bei Aufruf und stoppen, wenn keine weiteren Aufgaben auszuführen sind. Ein Aufruf der Funktion erstellt einen separaten Container (oder eine Sandbox) für die Verarbeitung der Aufrufe, es sei denn, ein vorhandener Container kann erneut verwendet werden. Daten, die an die Funktion gesendet werden, können von jedem der Container abgerufen werden.

Es können mehrere Aufrufe einer On-Demand-Funktion parallel ausgeführt werden.

Variablen und die Vorverarbeitungslogik, die außerhalb des Funktionshandlers definiert sind, werden nicht beibehalten, wenn neue Container erstellt werden.

- Langlebige (oder fixierte) Funktionen werden automatisch gestartet, wenn der AWS IoT Greengrass Kern gestartet und in einem einzigen Container ausgeführt wird. Alle Daten, die an die Funktion gesendet werden, können vom selben Container abgerufen werden.

Mehrere Aufrufe werden so lange in eine Warteschlange verschoben, bis frühere Aufrufe ausgeführt werden.

Variablen und die Vorverarbeitungslogik, die außerhalb des Funktionshandlers definiert sind, werden für jeden Aufruf des Funktionshandlers beibehalten.

Langlebige Lambda-Funktionen sind nützlich, wenn Sie ohne anfängliche Eingaben mit der Arbeit beginnen müssen. Eine langlebige Funktion kann beispielsweise ein ML-Modell laden und mit dessen Verarbeitung beginnen, um für den Empfang von Gerätedaten bereit zu sein.

Note

Denken Sie daran, dass die Zeitüberschreitungen von langlebigen Funktionen den Aufrufen ihrer Handler zugeordnet sind. Wenn Sie unbegrenzt laufenden Code ausführen möchten, muss er außerhalb des Handlers gestartet werden. Stellen Sie sicher, dass kein blockierender Code außerhalb des Handlers vorhanden ist, der möglicherweise verhindert, dass die Funktion den Initialisierungsvorgang abschließt.

Diese Funktionen werden ausgeführt, es sei denn, der Core stoppt (z. B. während einer Gruppenbereitstellung oder eines Gerätereustarts) oder die Funktion wechselt in einen Fehlerstatus (z. B. ein Handler-Timeout, eine nicht abgefangene Ausnahme oder wenn sie ihre Speichergrenzen überschreitet).

Weitere Informationen zur Wiederverwendung von Containern finden Sie unter [Grundlegendes zur Wiederverwendung von Containern AWS Lambda im](#) AWS Compute-Blog.

Ausführbare Lambda-Dateien

Diese Funktion ist für AWS IoT Greengrass Core v1.6 und höher verfügbar.

Eine ausführbare Lambda-Datei ist eine Art von Greengrass-Lambda-Funktion, mit der Sie Binärcode in der Kernumgebung ausführen können. Damit können Sie gerätespezifische Funktionen nativ ausführen und vom geringeren Platzbedarf des kompilierten Codes profitieren. Ausführbare Lambda-Dateien können durch Ereignisse aufgerufen werden, andere Funktionen aufrufen und auf lokale Ressourcen zugreifen.

Ausführbare Lambda-Dateien unterstützen nur den binären Kodierungstyp (nicht JSON), aber ansonsten können Sie sie in Ihrer Greengrass-Gruppe verwalten und wie andere Greengrass-Lambda-Funktionen bereitstellen. Der Prozess der Erstellung von ausführbaren Lambda-Dateien unterscheidet sich jedoch von der Erstellung von Python-, Java- und Node.js Lambda-Funktionen:

- Sie können die AWS Lambda Konsole nicht verwenden, um eine Lambda-Programmdatei zu erstellen (oder zu verwalten). Sie können eine Lambda-Programmdatei nur mithilfe der AWS Lambda API erstellen.
- Sie laden den Funktionscode AWS Lambda als kompilierte ausführbare Datei hoch, die das [AWS IoT Greengrass Core-SDK für C](#) enthält.
- Sie geben den Namen der ausführbaren Datei als Funktionshandler an.

Ausführbare Lambda-Dateien müssen bestimmte Aufrufe und Programmiermuster in ihrem Funktionscode implementieren. Folgendes muss beispielsweise von der `main`-Methode behandelt werden:

- Rufen Sie `gg_global_init` auf, um interne globale Variablen von Greengrass zu initialisieren. Diese Funktion muss vor dem Erstellen von Threads und vor dem Aufrufen anderer AWS IoT Greengrass Core-SDK-Funktionen aufgerufen werden.
- Rufen Sie `gg_runtime_start` auf, um den Funktionshandler bei der Greengrass Lambda-Laufzeit zu registrieren. Diese Funktion muss während der Initialisierung aufgerufen werden. Durch den Aufruf dieser Funktion wird der aktuelle Thread von der Laufzeit verwendet. Der optionale `GG_RT_OPT_ASYNC`-Parameter weist diese Funktion an, nicht zu blockieren, sondern ein neues Threads für die Laufzeit zu erstellen. Diese Funktion verwendet einen `SIGTERM`-Handler.

Der folgende Ausschnitt ist die `main` Methode aus dem Codebeispiel [simple_handler.c](#). [GitHub](#)

```
int main() {
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

[Weitere Informationen zu Anforderungen, Einschränkungen und anderen Implementierungsdetails finden Sie unter Core SDK for C.AWS IoT Greengrass](#)

Eine ausführbare Lambda-Datei erstellen

Nachdem Sie Ihren Code zusammen mit dem SDK kompiliert haben, verwenden Sie die AWS Lambda API, um eine Lambda-Funktion zu erstellen und Ihre kompilierte ausführbare Datei hochzuladen.

Note

Ihre Funktion muss mit einem C89-kompatiblen Compiler kompiliert werden.

Im folgenden Beispiel wird der CLI-Befehl [create-function](#) verwendet, um eine ausführbare Lambda-Datei zu erstellen. Der Befehl gibt folgendes an:

- Der Name der ausführbaren Datei für den Handler. Es muss der genaue Name Ihrer kompilierten ausführbaren Datei sein.
- Der Pfad zur `.zip`-Datei mit der kompilierten ausführbaren Datei.
- `arn:aws:greengrass:::runtime/function/executable` für die Laufzeit. Dies ist die Laufzeit für alle ausführbaren Lambda-Dateien.

 Note

Denn role Sie können den ARN einer beliebigen Lambda-Ausführungsrolle angeben. AWS IoT Greengrass verwendet diese Rolle nicht, aber der Parameter ist erforderlich, um die Funktion zu erstellen. Weitere Informationen zu Lambda-Ausführungsrollen finden Sie unter [AWS Lambda Permissions Model](#) im AWS Lambda Developer Guide.

```
aws lambda create-function \  
--region aws-region \  
--function-name function-name \  
--handler executable-name \  
--role role-arn \  
--zip-file fileb://file-name.zip \  
--runtime arn:aws:greengrass::runtime/function/executable
```

Verwenden Sie als Nächstes die AWS Lambda API, um eine Version zu veröffentlichen und einen Alias zu erstellen.

- Verwenden Sie [publish-version](#), um eine Funktionsversion zu veröffentlichen.

```
aws lambda publish-version \  
--function-name function-name \  
--region aws-region
```

- Verwenden Sie [create-alias](#), um einen Alias zu erstellen, der auf die soeben veröffentlichte Version weist. Wir empfehlen, dass Sie Lambda-Funktionen per Alias referenzieren, wenn Sie sie einer Greengrass-Gruppe hinzufügen.

```
aws lambda create-alias \  
--function-name function-name \  
--name alias-name \  
--function-version version-number \  
--region aws-region
```

 Note

Die AWS Lambda Konsole zeigt keine ausführbaren Lambda-Dateien an. Um den Funktionscode zu aktualisieren, müssen Sie die API verwenden. AWS Lambda

Fügen Sie dann die ausführbare Lambda-Datei zu einer Greengrass-Gruppe hinzu, konfigurieren Sie sie so, dass sie binäre Eingabedaten in ihren gruppenspezifischen Einstellungen akzeptiert, und stellen Sie die Gruppe bereit. Sie können dies in der AWS IoT Greengrass Konsole oder mithilfe der API tun. AWS IoT Greengrass

Wird AWS IoT Greengrass in einem Docker-Container ausgeführt

AWS IoT Greengrass [kann für die Ausführung in einem Docker-Container konfiguriert werden](#).

Sie können ein Dockerfile [über Amazon](#) herunterladen, auf dem CloudFront die AWS IoT Greengrass Core-Software und die zugehörigen Abhängigkeiten installiert sind. Informationen dazu, wie Sie das Docker-Image zur Ausführung auf verschiedenen Plattformarchitekturen modifizieren oder die Größe des Docker-Images verringern, finden Sie in der README-Datei im Docker-Paketdownload.

Um Ihnen den Einstieg in das Experimentieren zu erleichtern AWS IoT Greengrass, bietet es AWS auch vorgefertigte Docker-Images, auf denen die AWS IoT Greengrass Core-Software und die zugehörigen Abhängigkeiten installiert sind. Sie können ein Image von [Docker Hub](#) oder [Amazon Elastic Container Registry](#) (Amazon ECR) herunterladen. Diese vorgefertigten Images verwenden Amazon Linux 2 (x86_64) und Alpine Linux (x86_64, ARMv7L oder) Basis-Images. AArch64

 Important

Am 30. Juni 2022 AWS IoT Greengrass wurde die Wartung für Docker-Images der AWS IoT Greengrass Core-Software v1.x eingestellt, die in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub veröffentlicht wurden. Sie können diese Docker-Images weiterhin bis zum 30. Juni 2023, also 1 Jahr nach Ende der Wartung, von Amazon ECR und Docker Hub herunterladen. Nach dem Ende der Wartung am 30. Juni 2022 erhalten die Docker-Images der AWS IoT Greengrass Core-Software v1.x jedoch keine Sicherheitspatches oder Bugfixes mehr. Wenn Sie einen Produktions-Workload ausführen, der von diesen Docker-Images abhängt, empfehlen wir Ihnen, Ihre eigenen Docker-

Images mithilfe der bereitgestellten Dockerfiles zu erstellen. AWS IoT Greengrass Weitere Informationen finden Sie unter [AWS IoT Greengrass Docker-Software](#).

In diesem Thema wird beschrieben, wie Sie das AWS IoT Greengrass Docker-Image von Amazon ECR herunterladen und auf einer Windows-, macOS- oder Linux-Plattform (x86_64) ausführen. Das Thema umfasst die folgenden Schritte:

1. [Holen Sie sich das AWS IoT Greengrass Container-Image von Amazon ECR](#)
2. [Erstellen und Konfigurieren von Greengrass-Gruppe und Core](#)
3. [AWS IoT Greengrass Lokal ausführen](#)
4. [Konfigurieren der "Kein Container"-Containerisierung für die Gruppe](#)
5. [Stellen Sie Lambda-Funktionen im Docker-Container bereit](#)
6. [\(Optional\) Stellen Sie Client-Geräte, die mit Greengrass interagieren, im Docker-Container bereit](#)

Die folgenden Funktionen werden nicht unterstützt, wenn Sie sie in einem Docker-Container ausführen AWS IoT Greengrass :

- [Konnektoren](#), die im Greengrass-Container-Modus ausgeführt werden. Um einen Konnektor in einem Docker-Container auszuführen, muss der Konnektor im Modus No container (Kein Container) ausgeführt werden. Informationen zum Suchen von Konnektoren, die den No Container (Kein Container)-Modus unterstützen, finden Sie unter [the section called "AWS-mitgelieferte Greengrass-Anschlüsse"](#). Einige dieser Konnektoren weisen einen Isolationsmodus-Parameter auf, den Sie auf No container (Kein Container) festlegen müssen.
- [Lokale Geräte- und Volume-Ressourcen](#) Ihre benutzerdefinierten Lambda-Funktionen, die im Docker-Container ausgeführt werden, müssen direkt auf Geräte und Volumes auf dem Core zugreifen.

Diese Funktionen werden nicht unterstützt, wenn die Lambda-Laufzeitumgebung für die Greengrass-Gruppe auf [Kein Container](#) gesetzt ist, was für die Ausführung AWS IoT Greengrass in einem Docker-Container erforderlich ist.

Voraussetzungen

Bevor Sie mit diesem Tutorial beginnen, müssen Sie Folgendes tun.

- Sie müssen die folgende Software und Versionen auf Ihrem Hostcomputer installieren, basierend auf der von Ihnen ausgewählten Version AWS Command Line Interface (AWS CLI).

AWS CLI version 2

- [Docker-Version](#) 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, wir empfehlen jedoch 18.09 oder höher.
- AWS CLI Version 2.0.0 oder höher.
 - Informationen zur Installation der AWS CLI Version 2 finden Sie unter [AWS CLI Version 2 installieren](#).
 - Informationen zur AWS CLI Konfiguration [von finden Sie unter Konfiguration von AWS CLI](#).

Note

Um auf einem Windows-Computer auf eine neuere AWS CLI Version 2 zu aktualisieren, müssen Sie den [MSI-Installationsvorgang](#) wiederholen.

AWS CLI version 1

- [Docker-Version](#) 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, wir empfehlen jedoch 18.09 oder höher.
- [Python-Version](#) 3.6 oder höher.
- [pip](#), Version 18.1 oder höher.
- AWS CLI Version 1.17.10 oder höher
 - Informationen zur Installation der AWS CLI Version 1 finden Sie unter [AWS CLI Version 1 installieren](#).
 - Informationen zur AWS CLI Konfiguration [von finden Sie unter Konfiguration von AWS CLI](#).
 - Führen Sie den folgenden Befehl aus, um auf die neueste AWS CLI Version von Version 1 zu aktualisieren.

```
pip install awscli --upgrade --user
```

Note

Wenn Sie die [MSI-Installation](#) der AWS CLI Version 1 unter Windows verwenden, beachten Sie Folgendes:

- Wenn bei der Installation AWS CLI von Version 1 Botocore nicht installiert werden kann, versuchen Sie es mit der [Python- und Pip-Installation](#).
 - Um auf eine neuere AWS CLI Version 1 zu aktualisieren, müssen Sie den MSI-Installationsvorgang wiederholen.
- Um auf Ressourcen von Amazon Elastic Container Registry (Amazon ECR) zugreifen zu können, müssen Sie die folgende Berechtigung erteilen.
 - Amazon ECR verlangt von Benutzern, die `ecr:GetAuthorizationToken` Erlaubnis über eine AWS Identity and Access Management (IAM-) Richtlinie zu erteilen, bevor sie sich bei einer Registrierung authentifizieren und Bilder aus einem Amazon ECR-Repository per Push oder Pull abrufen können. Weitere Informationen finden Sie unter [Amazon ECR Repository-Richtlinienbeispiele](#) und [Zugriff auf ein Amazon ECR-Repository](#) im Amazon Elastic Container Registry-Benutzerhandbuch.

Schritt 1: Holen Sie sich das AWS IoT Greengrass Container-Image von Amazon ECR

AWS stellt Docker-Images bereit, auf denen die AWS IoT Greengrass Core-Software installiert ist.

Warning

Ab Version 1.11.6 der AWS IoT Greengrass Core-Software enthalten die Greengrass Docker-Images Python 2.7 nicht mehr, da Python 2.7 end-of-life im Jahr 2020 erreicht wurde und keine Sicherheitsupdates mehr erhält. Wenn Sie sich für ein Update auf diese Docker-Images entscheiden, empfehlen wir Ihnen, zu überprüfen, ob Ihre Anwendungen mit den neuen Docker-Images funktionieren, bevor Sie die Updates auf Produktionsgeräten bereitstellen. Wenn Sie Python 2.7 für Ihre Anwendung benötigen, die ein Greengrass Docker-Image verwendet, können Sie das Greengrass Dockerfile so ändern, dass es Python 2.7 für Ihre Anwendung enthält.

Wählen Sie für die Schritte, die zeigen, wie Sie das latest Bild aus Amazon ECR abrufen können, Ihr Betriebssystem aus:

Abrufen des Container-Abbilds (Linux)

Führen Sie die folgenden Befehle auf Ihrem Computer-Terminal aus.

1. Melden Sie sich bei der AWS IoT Greengrass Registrierung in Amazon ECR an.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

Bei erfolgreicher Ausführung wird als Ausgabe Login Succeeded gedruckt.

2. Rufen Sie das AWS IoT Greengrass Container-Image ab.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

Das latest Image enthält die neueste stabile Version der AWS IoT Greengrass Core-Software, die auf einem Amazon Linux 2-Basis-Image installiert ist. Sie können auch andere Images per Pull aus dem Repository abrufen. Um alle verfügbaren Bilder zu finden, überprüfen Sie die Seite Tags im [Docker Hub](https://hub.docker.com/r/aws-iot-greengrass) oder verwenden Sie den aws ecr list-images-Befehl. Beispiel:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --repository-name aws-iot-greengrass
```

3. Aktivieren Sie den symlink- und hardlink-Schutz. Wenn Sie mit der Ausführung AWS IoT Greengrass in einem Container experimentieren, können Sie die Einstellungen nur für den aktuellen Start aktivieren.

Note

Sie müssen diese Befehle möglicherweise mit sudo ausführen.

- Um die Einstellungen nur für den aktuellen Systemstart zu aktivieren:

```
echo 1 > /proc/sys/fs/protected_hardlinks
echo 1 > /proc/sys/fs/protected_symlinks
```

- So aktivieren Sie die Einstellungen, damit sie über Neustarts hinweg beibehalten werden:

```
echo '# AWS IoT Greengrass' >> /etc/sysctl.conf
```

```
echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf
echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf

sysctl -p
```

4. Aktivieren Sie die IPv4 Netzwerkweiterleitung, die für die AWS IoT Greengrass Cloud-Bereitstellung und die MQTT-Kommunikation unter Linux erforderlich ist. Legen Sie in der Datei `/etc/sysctl.conf` für `net.ipv4.ip_forward` "1" fest und laden Sie dann `sysctl`s neu.

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

Note

Sie können anstelle von nano einen Editor Ihrer Wahl verwenden.

Abrufen des Container-Abbilds (macOS)

Führen Sie die folgenden Befehle auf Ihrem Computer-Terminal aus.

1. Melden Sie sich bei der AWS IoT Greengrass Registrierung in Amazon ECR an.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

Bei erfolgreicher Ausführung wird als Ausgabe `Login Succeeded` gedruckt.

2. Rufen Sie das AWS IoT Greengrass Container-Image ab.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

Das `latest` Image enthält die neueste stabile Version der AWS IoT Greengrass Core-Software, die auf einem Amazon Linux 2-Basis-Image installiert ist. Sie können auch andere Images per Pull aus dem Repository abrufen. Um alle verfügbaren Bilder zu finden, überprüfen Sie die Seite Tags im [Docker Hub](https://hub.docker.com/_/aws-iot-greengrass) oder verwenden Sie den `aws ecr list-images`-Befehl. Beispiel:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Abrufen des Container-Abbilds (Windows)

Führen Sie in einem Eingabeaufforderungsfenster die folgenden Befehle aus. Um Docker-Befehle auf Windows verwenden zu können, muss der Docker-Desktop ausgeführt werden.

1. Melden Sie sich bei der AWS IoT Greengrass Registrierung in Amazon ECR an.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --  
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

Bei erfolgreicher Ausführung wird als Ausgabe `Login Succeeded` gedruckt.

2. Rufen Sie das AWS IoT Greengrass Container-Image ab.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Note

Das `latest` Image enthält die neueste stabile Version der AWS IoT Greengrass Core-Software, die auf einem Amazon Linux 2-Basis-Image installiert ist. Sie können auch andere Images per Pull aus dem Repository abrufen. Um alle verfügbaren Bilder zu finden, überprüfen Sie die Seite [Tags](#) im [Docker Hub](#) oder verwenden Sie den `aws ecr list-images`-Befehl. Beispiel:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Schritt 2: Erstellen und Konfigurieren von Greengrass-Gruppe und Core

Auf dem Docker-Image ist die AWS IoT Greengrass Core-Software installiert, aber Sie müssen eine Greengrass-Gruppe und einen Kern erstellen. Dazu gehören das Herunterladen von Zertifikaten und von der Core-Konfigurationsdatei.

- Führen Sie die Schritte unter [the section called “Modul 2: Installation der AWS IoT Greengrass Core-Software”](#) aus. Überspringen Sie die Schritte, in denen Sie die AWS IoT Greengrass Core-Software herunterladen und ausführen. Die Software und ihre Laufzeitabhängigkeiten sind im Docker-Image bereits eingerichtet.

Schritt 3: AWS IoT Greengrass Lokal ausführen

Nachdem Ihre Gruppe konfiguriert wurde, können Sie den Core konfigurieren und starten. Schrittweise Anleitungen hierzu erhalten Sie bei Auswahl Ihres Betriebssystems:

Lokales Ausführen von Greengrass (Linux)

Führen Sie die folgenden Befehle auf Ihrem Computer-Terminal aus.

- Erstellen Sie einen Ordner für die Sicherheitsressourcen des Geräts und verschieben Sie das Zertifikat und die Schlüssel in diesen Ordner. Führen Sie die folgenden Befehle aus. *path-to-security-files* Ersetzen Sie es durch den Pfad zu den Sicherheitsressourcen und *certificateId* durch die Zertifikat-ID in den Dateinamen.

```
mkdir /tmp/certs
mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
mv path-to-security-files/certificateId-public.pem.key /tmp/certs
mv path-to-security-files/certificateId-private.pem.key /tmp/certs
mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
```

- Erstellen Sie einen Ordner für die Konfiguration des Geräts und verschieben Sie die AWS IoT Greengrass Core-Konfigurationsdatei in diesen Ordner. Führen Sie die folgenden Befehle aus. *path-to-config-file* Ersetzen Sie durch den Pfad zur Konfigurationsdatei.

```
mkdir /tmp/config
mv path-to-config-file/config.json /tmp/config
```

- Starten Sie die Zertifikate AWS IoT Greengrass und die Konfigurationsdatei und binden Sie sie im Docker-Container ein.

Ersetzen Sie */tmp* durch den Pfad, in den Sie Ihre Zertifikate und Konfigurationsdatei dekomprimiert haben.

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
```

```
-v /tmp/certs:/greengrass/certs \  
-v /tmp/config:/greengrass/config \  
-p 8883:8883 \  
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
Setting up greengrass daemon  
Validating hardlink/softlink protection  
Waiting for up to 30s for Daemon to start  
  
Greengrass successfully started with PID: 10
```

Lokales Ausführen von Greengrass (macOS)

Führen Sie die folgenden Befehle auf Ihrem Computer-Terminal aus.

1. Erstellen Sie einen Ordner für die Sicherheitsressourcen des Geräts und verschieben Sie das Zertifikat und die Schlüssel in diesen Ordner. Führen Sie die folgenden Befehle aus. *path-to-security-files* Ersetzen Sie es durch den Pfad zu den Sicherheitsressourcen und *certificateId* durch die Zertifikat-ID in den Dateinamen.

```
mkdir /tmp/certs  
mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs  
mv path-to-security-files/certificateId-public.pem.key /tmp/certs  
mv path-to-security-files/certificateId-private.pem.key /tmp/certs  
mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
```

2. Erstellen Sie einen Ordner für die Konfiguration des Geräts und verschieben Sie die AWS IoT Greengrass Core-Konfigurationsdatei in diesen Ordner. Führen Sie die folgenden Befehle aus. *path-to-config-file* Ersetzen Sie durch den Pfad zur Konfigurationsdatei.

```
mkdir /tmp/config  
mv path-to-config-file/config.json /tmp/config
```

3. Starten Sie die Zertifikate AWS IoT Greengrass und die Konfigurationsdatei und binden Sie sie im Docker-Container ein.

Ersetzen Sie `/tmp` durch den Pfad, in den Sie Ihre Zertifikate und Konfigurationsdatei dekomprimiert haben.

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

Lokales Ausführen von Greengrass (Windows)

1. Erstellen Sie einen Ordner für die Sicherheitsressourcen des Geräts und verschieben Sie das Zertifikat und die Schlüssel in diesen Ordner. Führen Sie in einem Eingabeaufforderungsfenster die folgenden Befehle aus. *path-to-security-files* Ersetzen Sie es durch den Pfad zu den Sicherheitsressourcen und *certificateId* durch die Zertifikat-ID in den Dateinamen.

```
mkdir C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
```

2. Erstellen Sie einen Ordner für die Konfiguration des Geräts und verschieben Sie die AWS IoT Greengrass Core-Konfigurationsdatei in diesen Ordner. Führen Sie in einem Eingabeaufforderungsfenster die folgenden Befehle aus. *path-to-config-file* Ersetzen Sie durch den Pfad zur Konfigurationsdatei.

```
mkdir C:\Users\%USERNAME%\Downloads\config
move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
```

3. Starten Sie die Zertifikate AWS IoT Greengrass und die Konfigurationsdatei und binden Sie sie im Docker-Container ein. Führen Sie in einem Eingabeaufforderungsfenster die folgenden Befehle aus.

```
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-  
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs  
-v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883  
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Geben Sie bei einer entsprechenden Docker-Eingabeaufforderung Ihr C:\-Laufwerk für den Docker-Daemon frei, sodass er das Verzeichnis C:\ innerhalb des Docker-Containers mit einem Bind-Mount bereitstellen kann. Weitere Informationen finden Sie in der Docker-Dokumentation unter [Freigegebene Laufwerke](#).

Die Ausgabe sollte in etwa wie folgt aussehen:

```
Setting up greengrass daemon  
Validating hardlink/softlink protection  
Waiting for up to 30s for Daemon to start  
  
Greengrass successfully started with PID: 10
```

Note

Wenn der Container die Shell nicht öffnet und sofort beendet, können Sie das Problem debuggen, indem Sie ein Bind-Mount der Greengrass Runtime-Protokolle beim Start des Abbilds erstellen. Weitere Informationen finden Sie unter [the section called “Erhalten der Greengrass-Laufzeitprotokolle außerhalb des Docker-Containers”](#).

Schritt 4: Konfigurieren der "Kein Container"-Containerisierung für die Greengrass-Gruppe

Wenn Sie AWS IoT Greengrass in einem Docker-Container ausführen, müssen alle Lambda-Funktionen ohne Containerisierung ausgeführt werden. In diesem Schritt legen Sie als Standard-Containerisierung für die Gruppe No container (Kein Container) fest. Dies muss vor der erstmaligen Bereitstellung der Gruppe geschehen.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe aus, deren Einstellungen Sie ändern möchten.
3. Wählen Sie die Registerkarte Lambda-Funktionen.
4. Wählen Sie unter Standard-Laufzeitumgebung für Lambda-Funktionen die Option Bearbeiten aus.
5. Ändern Sie in der Laufzeitumgebung Standard-Lambda-Funktion bearbeiten unter Standardcontainerisierung für Lambda-Funktionen die Containerisierungseinstellungen.
6. Wählen Sie Speichern.

Die Änderungen werden wirksam, wenn die Gruppe bereitgestellt wird.

Weitere Informationen finden Sie unter [the section called “Standardcontainerisierung für Lambda-Funktionen in einer Gruppe festlegen”](#).

 Note

Standardmäßig verwenden Lambda-Funktionen die Gruppencontainerisierungseinstellung. Wenn Sie die Einstellung Kein Container für irgendwelche Lambda-Funktionen überschreiben, wenn sie AWS IoT Greengrass in einem Docker-Container ausgeführt werden, schlägt die Bereitstellung fehl.

Schritt 5: Lambda-Funktionen im AWS IoT Greengrass Docker-Container bereitstellen

Sie können langlebige Lambda-Funktionen im Greengrass Docker-Container bereitstellen.

- Folgen Sie den Schritten unter [the section called “Modul 3 \(Teil 1\): Lambda-Funktionen auf AWS IoT Greengrass”](#), um eine langlebige Hello World Lambda-Funktion für den Container bereitzustellen.

Schritt 6: (Optional) Stellen Sie Client-Geräte bereit, die mit Greengrass interagieren, das im Docker-Container ausgeführt wird

Sie können auch Client-Geräte bereitstellen, die mit der Ausführung interagieren, AWS IoT Greengrass wenn es in einem Docker-Container ausgeführt wird.

- Folgen Sie den Schritten unter [the section called “Modul 4: Interaktion mit Client-Geräten in einer AWS IoT Greengrass Gruppe”](#), um Client-Geräte bereitzustellen, die eine Verbindung zum Core herstellen und MQTT-Nachrichten senden.

Den AWS IoT Greengrass Docker-Container stoppen

Um den AWS IoT Greengrass Docker-Container zu stoppen, drücken Sie Strg+C in Ihrem Terminal oder in der Befehlszeile. Diese Aktion wird SIGTERM an den Greengrass-Daemon-Prozess gesendet, um den Greengrass-Daemon-Prozess und alle Lambda-Prozesse, die vom Daemon-Prozess gestartet wurden, herunterzufahren. Der Docker-Container wird mit dem Prozess `/dev/init` als PID 1 initialisiert, wodurch alle verbliebenen Zombieprozesse entfernt werden. Weitere Informationen finden Sie in der [Referenz zu docker run](#).

Problembehandlung in einem Docker-Container AWS IoT Greengrass

Verwenden Sie die folgenden Informationen, um Probleme bei der Ausführung AWS IoT Greengrass in einem Docker-Container zu beheben.

Fehler: Eine interaktive Anmeldung kann nicht von einem Gerät aus durchgeführt werden, das kein TTY-Gerät ist.

Lösung: Dieser Fehler kann auftreten, wenn Sie den Befehl `aws ecr get-login-password` ausführen. Stellen Sie sicher, dass Sie die neueste AWS CLI Version 2 oder Version 1 installiert haben. Wir empfehlen Ihnen, die AWS CLI Version 2 zu verwenden. Weitere Informationen finden Sie unter [Installieren der AWS CLI](#) im AWS Command Line Interface -Benutzerhandbuch.

Fehler: Unbekannte Optionen: `-no-include-email`.

Lösung: Dieser Fehler kann auftreten, wenn Sie den Befehl `aws ecr get-login` ausführen. Vergewissern Sie sich, dass Sie die neueste AWS CLI Version installiert haben (zum Beispiel `run: pip install awscli --upgrade --user`). Wenn Sie Windows verwenden und die CLI mit

dem MSI-Installationsprogramm installiert haben, müssen Sie den Installationsvorgang wiederholen. Weitere Informationen finden Sie im AWS Command Line Interface Benutzerhandbuch AWS Command Line Interface [unter Installation von unter Microsoft Windows](#).

Warnung: IPv4 ist deaktiviert. Das Netzwerk wird nicht funktionieren.

Lösung: Möglicherweise erhalten Sie diese oder eine ähnliche Meldung, wenn Sie AWS IoT Greengrass auf einem Linux-Computer arbeiten. Aktivieren Sie die IPv4 Netzwerkweiterleitung wie in diesem [Schritt](#) beschrieben. AWS IoT Greengrass Cloud-Bereitstellung und MQTT-Kommunikation funktionieren nicht, wenn die IPv4 Weiterleitung nicht aktiviert ist. Weitere Informationen finden Sie unter [Configure namespaced kernel parameters \(sysctls\) at runtime](#) in der Docker-Dokumentation.

Fehler: Eine Firewall blockiert die Freigabe von Dateien zwischen Fenstern und den Containern.

Lösung: Sie können diese Fehlermeldung oder eine Firewall Detected-Nachricht erhalten, wenn Sie Docker auf einem Windows-Computer ausführen. Dies kann auch auftreten, wenn Sie an einem Virtual Private Network (VPN) angemeldet sind und Ihre Netzwerkeinstellungen die Bereitstellung des freigegebenen Laufwerks verhindern. Deaktivieren Sie in diesem Fall das VPN und führen Sie den Docker-Container erneut aus.

Fehler: Beim Aufrufen der GetAuthorizationToken Operation ist ein Fehler aufgetreten (AccessDeniedException): User: arn:aws:iam: :user/ <account-id>ist <user-name>nicht berechtigt,: ecr: on resource: * auszuführen GetAuthorizationToken

Möglicherweise erhalten Sie diesen Fehler, wenn Sie den `aws ecr get-login-password` Befehl ausführen, wenn Sie nicht über ausreichende Berechtigungen für den Zugriff auf ein Amazon ECR-Repository verfügen. Weitere Informationen finden Sie unter [Beispiele für Amazon ECR Repository-Richtlinien](#) und [Zugriff auf ein Amazon ECR-Repository](#) im Amazon ECR-Benutzerhandbuch.

Allgemeine Hilfe AWS IoT Greengrass zur Fehlerbehebung finden Sie unter. [Fehlerbehebung](#)

Debuggen AWS IoT Greengrass in einem Docker-Container

Zum Debuggen von Problemen mit einem Docker-Container können Sie die Greengrass-Laufzeitprotokolle erhalten oder eine interaktive Shell an den Docker-Container anfügen.

Erhalten der Greengrass-Laufzeitprotokolle außerhalb des Docker-Containers

Sie können den AWS IoT Greengrass Docker-Container ausführen, nachdem Sie das Verzeichnis per Bind-Mounten bereitgestellt haben. /greengrass/ggc/var/log Die Protokolle bleiben auch dann erhalten, wenn der Container beendet oder entfernt wird.

Unter Linux oder macOS

[Beenden Sie alle Greengrass-Docker-Container](#), die auf dem Host ausgeführt werden, und führen Sie dann den folgenden Befehl in einem Terminal aus. Auf diese Weise wird ein Bind-Mount des Greengrass log-Verzeichnisses erstellt und das Docker-Image gestartet.

Ersetzen Sie /tmp durch den Pfad, in den Sie Ihre Zertifikate und Konfigurationsdatei dekomprimiert haben.

```
docker run --rm --init -it --name aws-iot-greengrass \
  --entrypoint /greengrass-entrypoint.sh \
  -v /tmp/certs:/greengrass/certs \
  -v /tmp/config:/greengrass/config \
  -v /tmp/log:/greengrass/ggc/var/log \
  -p 8883:8883 \
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Anschließend können Sie Ihre Protokolle auf Ihrem Host unter /tmp/log überprüfen, um zu sehen, welche Entwicklungen in Greengrass während der Ausführung des Docker-Containers eingetreten sind.

Unter Windows

[Beenden Sie alle Greengrass-Docker-Container](#), die auf dem Host ausgeführt werden, und führen Sie dann den folgenden Befehl in einem Befehlszeilenfenster aus. Auf diese Weise wird ein Bind-Mount des Greengrass log-Verzeichnisses erstellt und das Docker-Image gestartet.

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-
west-2.amazonaws.com/aws-iot-greengrass:latest
```

Anschließend können Sie Ihre Protokolle auf Ihrem Host unter `C:/Users/%USERNAME%/Downloads/log` überprüfen, um zu sehen, welche Entwicklungen in Greengrass während der Ausführung des Docker-Containers eingetreten sind.

Anfügen einer interaktiven Shell an den Docker-Container

Sie können eine interaktive Shell an einen laufenden AWS IoT Greengrass Docker-Container anhängen. Auf diese Weise können Sie den Status des Greengrass Docker-Containers untersuchen.

Unter Linux oder macOS

Führen Sie den folgenden Befehl in einem separaten Terminal aus, während der Greengrass-Docker-Container ausgeführt wird.

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

Unter Windows

Führen Sie die folgenden Befehle in einem separaten Befehlszeilenfenster aus, während der Greengrass-Docker-Container ausgeführt wird.

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

gg-container-id Ersetzen Sie es durch das `container_id` Ergebnis des vorherigen Befehls.

```
docker exec -it gg-container-id /bin/bash
```

Greifen Sie mit Lambda-Funktionen und Konnektoren auf lokale Ressourcen zu

Diese Funktion ist für AWS IoT Greengrass Core v1.3 und höher verfügbar.

Mit AWS IoT Greengrass können Sie AWS Lambda Funktionen erstellen und [Konnektoren](#) in der Cloud konfigurieren und sie zur lokalen Ausführung auf Kerngeräten bereitstellen. Auf Greengrass-Kernen, auf denen Linux ausgeführt wird, können diese lokal bereitgestellten Lambda-Funktionen und -Konnektoren auf lokale Ressourcen zugreifen, die physisch auf dem Greengrass-Core-Gerät vorhanden sind. Um beispielsweise mit Geräten zu kommunizieren, die über Modbus verbunden sind CANbus, oder Sie können Ihre Lambda-Funktion aktivieren, um auf die serielle Schnittstelle des Kerngeräts zuzugreifen. Um den sicheren Zugriff auf lokale Ressourcen zu konfigurieren, müssen Sie die Sicherheit der physischen Hardware sowie des Betriebssystems des Greengrass Core-Geräts gewährleisten.

Die ersten Schritte zum Zugriff auf lokale Ressourcen finden Sie in den folgenden Tutorials:

- [So konfigurieren Sie den lokalen Ressourcenzugriff über die AWS Befehlszeilenschnittstelle](#)
- [So konfigurieren Sie den lokalen Ressourcenzugriff mit dem AWS-Managementkonsole](#)

Unterstützte Ressourcentypen

Sie können auf zwei Arten lokaler Ressourcen zugreifen: Volume-Ressourcen und Geräteressourcen.

Volume-Ressourcen

Dateien und Verzeichnisse auf dem Stammdateisystem (außer unter `/sys`, `/dev` oder `/var`). Dazu zählen:

- Ordner oder Dateien, die zum Lesen oder Schreiben von Informationen über die Funktionen von Greengrass Lambda verwendet werden (z. B. `/usr/lib/python2.x/site-packages/local`).
- Ordner oder Dateien unter dem `/proc`-Dateisystem des Hosts (z. B. `/proc/net` oder `/proc/stat`). Wird in Version 1.6 oder höher unterstützt. Zusätzliche Anforderungen finden Sie unter [the section called "Volume-Ressourcen im /proc-Verzeichnis"](#).

 Tip

Um die Verzeichnisse `/var`, `/var/run` und `/var/lib` als Volume-Ressourcen zu konfigurieren, mounten Sie zunächst das Verzeichnis in einen anderen Ordner und konfigurieren diesen Ordner als Volume-Ressource.

Wenn Sie Volume-Ressourcen konfigurieren, geben Sie einen Pfad für `source` (Quelle) und einen Pfad für `destination` (Ziel) an. Der Quellpfad ist der absolute Pfad der Ressource auf dem Host. Der Zielpfad ist der absolute Pfad der Ressource innerhalb der Lambda-Namespace-Umgebung. Dies ist der Container, in dem eine Greengrass Lambda-Funktion oder ein Connector ausgeführt wird. Änderungen am Zielpfad werden im Quellpfad des Dateisystems auf dem Host-Dateisystem wiedergegeben.

 Note

Dateien im Zielpfad sind nur im Lambda-Namespace sichtbar. Sie können sie in einem regulären Linux-Namespace nicht sehen.

Geräteressourcen

Dateien unter `/dev`. Nur zeichenorientierte Geräte oder Blockgeräte unter `/dev` sind als Geräteressourcen zulässig. Dazu zählen:

- Serielle Ports, die für die Kommunikation mit über serielle Ports angeschlossenen Geräten verwendet werden (zum Beispiel `/dev/ttyS0`, `/dev/ttyS1`).
- USB, die zum Anschließen von USB-Peripheriegeräten verwendet werden (zum Beispiel. `/dev/ttyUSB0` oder `/dev/bus/usb`).
- GPIOs wird für Sensoren und Aktoren über GPIO verwendet (z. B.). `/dev/gpiomem`
- GPUs wird verwendet, um maschinelles Lernen mithilfe von On-Board zu beschleunigen GPUs (z. B.). `/dev/nvidia0`
- Kameras, mit denen Bilder und Videos erfasst werden (zum Beispiel `/dev/video0`).

 Note

`/dev/shm` ist eine Ausnahme. Sie kann nur als eine Volume-Ressource konfiguriert werden. Ressourcen unter `/dev/shm` muss die Berechtigung `rw` erteilt werden.

AWS IoT Greengrass unterstützt auch Ressourcentypen, die zur Durchführung von Inferenzen für maschinelles Lernen verwendet werden. Weitere Informationen finden Sie unter [Durchführen von Machine Learning-Inferenzen](#).

Voraussetzungen

Die folgenden Anforderungen gelten für die Konfiguration des sicheren Zugriffs auf lokale Ressourcen:

- Sie müssen AWS IoT Greengrass Core Software v1.3 oder höher verwenden. Um Ressourcen für das Verzeichnis `/proc` des Hosts zu erstellen, müssen Sie Version 1.6 oder höher verwenden.
- Die lokale Ressource (einschließlich aller erforderlichen Treiber und Bibliotheken) muss ordnungsgemäß auf dem Greengrass Core-Gerät installiert sein und während der gesamten Nutzung zur Verfügung stehen.
- Die gewünschte Operation der Ressource und der Zugriff auf die Ressource dürfen keine Root-Berechtigungen erfordern.
- Nur die Berechtigung `read` oder `read` and `write` ist verfügbar. Lambdafunktionen können keine privilegierten Operationen für die Ressourcen ausführen.
- Sie müssen den vollständigen Pfad der lokalen Ressource im Betriebssystem des Greengrass Core-Geräts bereitstellen.
- Der Name oder die ID einer Ressource kann maximal 128 Zeichen umfassen und muss dem Muster `[a-zA-Z0-9:_-]+` entsprechen.

Volume-Ressourcen im `/proc`-Verzeichnis

Die folgenden Überlegungen gelten für Volume-Ressourcen im `/proc`-Verzeichnis des Hosts.

- Sie müssen AWS IoT Greengrass Core Software v1.6 oder höher verwenden.
- Sie können nur Lesezugriff für Lambda-Funktionen zulassen, aber keinen Lese- und Schreibzugriff. Diese Zugriffsebene wird verwaltet von AWS IoT Greengrass

- Möglicherweise müssen Sie auch Betriebssystemgruppen Berechtigungen erteilen, um Lesezugriff im Dateisystem zu aktivieren. Nehmen wir beispielsweise an, dass Ihr Quellverzeichnis oder Ihre Quelldatei über die Dateiberechtigung „660“ verfügt. Dies bedeutet, dass nur der Eigentümer bzw. der Benutzer in der Gruppe Lese- (oder Schreib)-Zugriff darauf hat. In diesem Fall müssen Sie die Berechtigungen des Besitzers der Betriebssystemgruppe der Ressource hinzufügen. Weitere Informationen finden Sie unter [the section called “Dateizugriffsberechtigung des Gruppenbesitzers”](#).
- Die Host-Umgebung und der Lambda-Namespace enthalten beide ein /proc-Verzeichnis. Achten Sie also darauf, Namenskonflikte zu vermeiden, wenn Sie den Zielpfad angeben. Wenn /proc beispielsweise der Quellpfad ist, können Sie /host-proc als Zielpfad angeben (oder einen beliebigen anderen Namen anstelle von „/proc“).

Dateizugriffsberechtigung des Gruppenbesitzers

Ein AWS IoT Greengrass Lambda-Funktionsprozess läuft normalerweise als `ggc_user` und `ggc_group`. Sie können dem Lambda-Funktionsprozess in der lokalen Ressourcendefinition jedoch wie folgt zusätzliche Dateizugriffsberechtigungen erteilen:

- Um die Berechtigungen der Linux-Gruppe hinzuzufügen, der die Ressource gehört, verwenden Sie den `GroupOwnerSetting#AutoAddGroupOwner` Parameter oder die Option `Dateisystemberechtigungen` automatisch hinzufügen der Systemgruppe, der die Ressource gehört, gehört.
- Um die Berechtigungen einer anderen Linux-Gruppe hinzuzufügen, verwenden Sie den `GroupOwnerSetting#GroupOwner` Parameter oder die Konsolenoption `Geben Sie eine andere Systemgruppe an, um Dateisystemberechtigungen hinzuzufügen`. Der Wert `GroupOwner` wird ignoriert, wenn `GroupOwnerSetting#AutoAddGroupOwner` auf `"true"` gesetzt ist.

Ein AWS IoT Greengrass Lambda-Funktionsprozess erbt alle Dateisystemberechtigungen von `ggc_user` `ggc_group`, und der Linux-Gruppe (falls hinzugefügt). Damit die Lambda-Funktion auf eine Ressource zugreifen kann, muss der Lambda-Funktionsprozess über die erforderlichen Berechtigungen für die Ressource verfügen. Sie können die Berechtigung der Ressource gegebenenfalls mit dem `chmod(1)`-Befehl ändern.

Weitere Informationen finden Sie auch unter

- [Servicekontingente](#) für Ressourcen in der Allgemeine Amazon Web Services-Referenz

So konfigurieren Sie den lokalen Ressourcenzugriff über die AWS Befehlszeilenschnittstelle

Diese Funktion ist für AWS IoT Greengrass Core v1.3 und höher verfügbar.

Um eine lokale Ressource zu verwenden, müssen Sie eine Ressourcendefinition zur Gruppendefinition hinzufügen, die auf Ihrem Greengrass Core-Gerät bereitgestellt wird. Die Gruppendefinition muss ebenfalls eine Lambda-Funktionsdefinition enthalten, in der Sie Ihren Lambda-Funktionen Zugriffsberechtigungen für lokale Ressourcen erteilen. Weitere Informationen einschließlich Anforderungen und Einschränkungen finden Sie unter [Greifen Sie mit Lambda-Funktionen und Konnektoren auf lokale Ressourcen zu](#).

Dieses Tutorial beschreibt den Prozess zum Erstellen einer lokalen Ressource und zum Konfigurieren des Zugriffs darauf mithilfe der AWS Command Line Interface (CLI). Sie müssen bereits eine Greengrass-Gruppe wie in [Erste Schritte mit AWS IoT Greengrass](#) beschrieben erstellt haben, um die Schritte im Tutorial auszuführen.

Ein Tutorial, das die verwendet AWS-Managementkonsole, finden Sie unter [So konfigurieren Sie den lokalen Ressourcenzugriff mit dem AWS-Managementkonsole](#).

Erstellen lokaler Ressourcen

Zuerst erstellen Sie eine Ressourcendefinition, die die Ressourcen festlegt, auf die mit dem Befehl [CreateResourceDefinition](#) zugegriffen wird. In diesem Beispiel erstellen wir die beiden Ressourcen TestDirectory und TestCamera:

```
aws greengrass create-resource-definition --cli-input-json '{
  "Name": "MyLocalVolumeResource",
  "InitialVersion": {
    "Resources": [
      {
        "Id": "data-volume",
        "Name": "TestDirectory",
        "ResourceDataContainer": {
          "LocalVolumeResourceData": {
            "SourcePath": "/src/LRAtest",
            "DestinationPath": "/dest/LRAtest",
            "GroupOwnerSetting": {
```

```

        "AutoAddGroupOwner": true,
        "GroupOwner": ""
    }
}
},
{
    "Id": "data-device",
    "Name": "TestCamera",
    "ResourceDataContainer": {
        "LocalDeviceResourceData": {
            "SourcePath": "/dev/video0",
            "GroupOwnerSetting": {
                "AutoAddGroupOwner": true,
                "GroupOwner": ""
            }
        }
    }
}
]
}
}'

```

Resources: eine Liste von Resource-Objekten in der Greengrass-Gruppe. Eine Greengrass-Gruppe kann bis zu 50 Ressourcen aufweisen.

Resource#Id: die eindeutige Kennung der Ressource. Die ID wird verwendet, um auf eine Ressource in der Konfiguration der Lambda-Funktion zu verweisen. Maximale Länge: 128 Zeichen. Muster: [a-zA-Z0-9:_-]+.

Resource#Name: der Name der Ressource. Die Ressourcename wird in der Greengrass-Konsole angezeigt. Maximale Länge: 128 Zeichen. Muster: [a-zA-Z0-9:_-]+.

LocalDeviceResourceData# SourcePath: Der lokale absolute Pfad der Geräteressource. Der Quellpfad für eine Geräteressource kann nur auf ein zeichenorientiertes Gerät oder Blockgerät unter /dev verweisen.

LocalVolumeResourceData# SourcePath: Der lokale absolute Pfad der Volume-Ressource auf dem Greengrass-Core-Gerät. Dieser Speicherort befindet sich außerhalb des [Containers](#), indem die Funktion ausgeführt wird. Der Quellpfad für einen Volume-Ressourcentyp darf nicht mit /sys beginnen.

LocalVolumeResourceData# DestinationPath: Der absolute Pfad der Volume-Ressource innerhalb der Lambda-Umgebung. Dieser Speicherort befindet sich innerhalb des Containers, indem die Funktion ausgeführt wird.

GroupOwnerSetting: Ermöglicht es Ihnen, zusätzliche Gruppenberechtigungen für den Lambda-Prozess zu konfigurieren. Dies ist ein optionales Feld. Weitere Informationen finden Sie unter [Dateizugriffsberechtigung des Gruppenbesitzers](#).

GroupOwnerSetting# AutoAddGroupOwner: Falls wahr, fügt Greengrass automatisch den angegebenen Linux-Betriebssystemgruppenbesitzer der Ressource zu den Lambda-Prozessberechtigungen hinzu. Auf diese Weise erhält der Lambda-Prozess die Dateizugriffsberechtigungen der hinzugefügten Linux-Gruppe.

GroupOwnerSetting# GroupOwner: Gibt den Namen der Linux-Betriebssystemgruppe an, deren Rechte dem Lambda-Prozess hinzugefügt wurden. Dies ist ein optionales Feld.

Der ARN einer Ressourcendefinitionsversion wird durch [CreateResourceDefinition](#) zurückgegeben. Der ARN sollte beim Aktualisieren einer Gruppendefinition verwendet werden.

```
{
  "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
  "Name": "MyLocalVolumeResource",
  "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
  "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
  "CreationTimestamp": "2017-11-15T01:18:42.153Z",
  "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
  "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

Erstellen der Greengrass-Funktion

Erstellen Sie nach der Erstellung der Ressourcen mithilfe des [CreateFunctionDefinition](#)-Befehls die Greengrass-Funktion und erteilen Sie der Funktion Zugriff auf die Ressource:

```
aws greengrass create-function-definition --cli-input-json '{
  "Name": "MyFunctionDefinition",
  "InitialVersion": {
    "Functions": [
```

```

    {
      "Id": "greengrassLraTest",
      "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
      "FunctionConfiguration": {
        "Pinned": false,
        "MemorySize": 16384,
        "Timeout": 30,
        "Environment": {
          "ResourceAccessPolicies": [
            {
              "ResourceId": "data-volume",
              "Permission": "rw"
            },
            {
              "ResourceId": "data-device",
              "Permission": "ro"
            }
          ],
          "AccessSysfs": true
        }
      }
    }
  ]
}
}'

```

ResourceAccessPolicies: Enthält die `resourceId` und `permission`, die der Lambda-Funktion Zugriff auf die Ressource gewähren. Eine Lambda-Funktion kann auf maximal 20 Ressourcen zugreifen.

ResourceAccessPolicy#Permission: Gibt an, welche Berechtigungen die Lambda-Funktion für die Ressource hat. Die verfügbaren Optionen sind `rw` (lesen/schreiben) oder `ro` (schreibgeschützt).

AccessSysfs: Falls wahr, kann der Lambda-Prozess Lesezugriff auf den `/sys` Ordner auf dem Greengrass-Core-Gerät haben. Dies wird in Fällen verwendet, in denen die Greengrass Lambda-Funktion Geräteinformationen lesen muss. `/sys`

Auch hier gibt [CreateFunctionDefinition](#) den ARN einer Funktionsdefinitionsversion zurück. Der ARN sollte in Ihrer Gruppenspezifischen Definition verwendet werden.

```
{
```

```

    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
    "Name": "MyFunctionDefinition",
    "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
    "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
    "CreationTimestamp": "2017-11-22T02:28:02.325Z",
    "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}

```

Fügen Sie die Lambda-Funktion zur Gruppe hinzu

Fügen Sie zum Schluss die Funktion mit [CreateGroupVersion](#) zur Gruppe hinzu. Beispiel:

```

aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-
caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbcb6/
versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"

```

Note

Wie Sie die Gruppen-ID abrufen, die mit diesem Befehl verwendet wird, erfahren Sie unter [the section called “Abrufen der Gruppen-ID”](#).

Es wird eine neue Gruppenversion zurückgegeben:

```

{
    "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/
b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
    "Version": "291917fb-ec54-4895-823e-27b52da25481",
}

```

```
"CreationTimestamp": "2017-11-22T01:47:22.487Z",
  "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

Ihre Greengrass-Gruppe enthält jetzt die `lraTest` Lambda-Funktion, die Zugriff auf zwei Ressourcen hat: `und`. `TestDirectory` `TestCamera`

Diese in Python geschriebene Lambda-Beispielfunktion `lraTest.py` schreibt in die lokale Volume-Ressource:

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRatest. Then it reads the file and
# publishes the content to the AWS IoT LRatest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRatest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

Diese Befehle werden von der Greengrass-API bereitgestellt, um Ressourcendefinitionen und Ressourcendefinitionsversionen zu erstellen und zu verwalten:

- [CreateResourceDefinition](#)
- [CreateResourceDefinitionVersion](#)
- [DeleteResourceDefinition](#)
- [GetResourceDefinition](#)
- [GetResourceDefinitionVersion](#)
- [ListResourceDefinitions](#)
- [ListResourceDefinitionVersions](#)
- [UpdateResourceDefinition](#)

Fehlerbehebung

- F: Warum schlägt meine Greengrass-Gruppenbereitstellung mit einer Fehlermeldung wie der folgenden fehl:

```
group config is invalid:
  ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
```

A: Dieser Fehler gibt an, dass der Lambda-Prozess nicht für den Zugriff auf die angegebene Ressource berechtigt ist. Die Lösung besteht darin, die Dateiberechtigung der Ressource zu ändern, sodass Lambda darauf zugreifen kann. (Für Einzelheiten vgl. [Dateizugriffsberechtigung des Gruppenbesitzers](#)).

- F: Warum startet die Lambda-Funktion nicht und gibt eine Fehlermeldung in "runtime.log" aus, wenn ich /var/run als Volume-Ressource konfiguriere?

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \"/var/run\" to rootfs \"/
greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/
rootfs_sys/run\"
caused \"invalid argument\""
```


A: AWS IoT Greengrass Core unterstützt derzeit nicht die Konfiguration von `/var/run`, und `/var/lib` als Volume-Ressourcen. Um dieses Problem zu umgehen, können Sie zuerst `/var`, `/var/run` oder `/var/lib` in einen anderen Ordner mounten und anschließend den Ordner als Volume-Ressource konfigurieren.

- F: Warum startet die Lambda-Funktion nicht und gibt eine Fehlermeldung in „runtime.log“ aus, wenn ich `/dev/shm` als Volume-Ressource mit Berechtigung für den schreibgeschützten Zugriff konfiguriere?

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda container.
container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \"\"/dev/shm\" to rootfs \"\"/greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"\"/greengrass/ggc/packages/1.3.0/rootfs_sys/dev/shm\" caused \"\"operation not permitted\"\"\""
```

A: `/dev/shm` kann nur mit Lese-/Schreibzugriff konfiguriert werden. Ändern Sie die Ressourcenberechtigung zu `rw`, um dieses Problem zu beheben.

So konfigurieren Sie den lokalen Ressourcenzugriff mit dem AWS-Managementkonsole

Diese Funktion ist für AWS IoT Greengrass Core v1.3 und höher verfügbar.

Sie können Lambda-Funktionen so konfigurieren, dass sie sicher auf lokale Ressourcen auf dem Greengrass-Host-Core-Gerät zugreifen. Zu lokalen Ressourcen zählen Busse und Peripheriegeräte, die sich physisch auf dem Host befinden, oder Dateisystemvolumes auf dem Hostbetriebssystem. Weitere Informationen einschließlich Anforderungen und Einschränkungen finden Sie unter [Greifen Sie mit Lambda-Funktionen und Konnektoren auf lokale Ressourcen zu](#).

In diesem Tutorial wird beschrieben, wie Sie mit der AWS-Managementkonsole Zugriff auf lokale Ressourcen konfigurieren, die auf einem AWS IoT Greengrass Core-Gerät vorhanden sind. Es enthält die folgenden allgemeinen Schritte:

1. [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#)
2. [Eine Lambda-Funktion erstellen und veröffentlichen](#)
3. [Fügen Sie die Lambda-Funktion zur Gruppe hinzu](#)

4. [Hinzufügen einer lokalen Ressource zur Gruppe](#)
5. [Hinzufügen von Abonnements zur Gruppe](#)
6. [Bereitstellen der Gruppe](#)

Ein Tutorial, das die verwendet AWS Command Line Interface, finden Sie unter [So konfigurieren Sie den lokalen Ressourcenzugriff über die AWS Befehlszeilenschnittstelle](#).

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Kern (v1.3 oder höher). Weitere Informationen zum Erstellen einer Greengrass-Gruppe oder eines Greengrass-Cores finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#).
- Die folgenden Verzeichnisse auf dem Greengrass Core-Gerät:
 - /src/ LRAtest
 - /dest/ LRAtest

Die Besitzergruppe dieser Verzeichnisse muss über Lese- und Schreibzugriff auf die Verzeichnisse verfügen. Sie können den folgenden Befehl verwenden, um Zugriff zu erteilen:

```
sudo chmod 0775 /src/LRAtest
```

Schritt 1: Erstellen Sie ein Lambda-Funktionsbereitstellungspaket

In diesem Schritt erstellen Sie ein Lambda-Funktionsbereitstellungspaket, bei dem es sich um eine ZIP-Datei handelt, die den Code und die Abhängigkeiten der Funktion enthält. Sie laden auch das AWS IoT Greengrass Core-SDK herunter, um es als Abhängigkeit in das Paket aufzunehmen.

1. Kopieren Sie auf Ihrem Computer folgendes Python-Skript in eine lokale Datei mit dem Namen `lraTest.py`. Dies ist die App-Logik für die Lambda-Funktion.

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.
```

```
import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
            client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

2. Laden Sie von der [AWS IoT Greengrass Core SDK-Downloadseite](#) das AWS IoT Greengrass Core SDK für Python auf Ihren Computer herunter.
3. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der greengrasssdk-Ordner.
4. Packen Sie die folgenden Elemente in einer ZIP-Datei mit dem Namen `lraTestLambda.zip`:
 - `lraTest.py`. App-Logik.
 - `greengrasssdk`. Erforderliche Bibliothek für alle Python-Lambda-Funktionen.

Die `lraTestLambda.zip` Datei ist Ihr Lambda-Funktionsbereitstellungspaket. Jetzt sind Sie bereit, eine Lambda-Funktion zu erstellen und das Bereitstellungspaket hochzuladen.

Schritt 2: Eine Lambda-Funktion erstellen und veröffentlichen

In diesem Schritt verwenden Sie die AWS Lambda Konsole, um eine Lambda-Funktion zu erstellen und sie für die Verwendung Ihres Bereitstellungspakets zu konfigurieren. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

Erstellen Sie zunächst die Lambda-Funktion.

1. Wählen Sie in der AWS-Managementkonsole Dienste aus und öffnen Sie die AWS Lambda Konsole.
2. Wählen Sie Funktionen.
3. Wählen Sie „Funktion erstellen“ und anschließend „Von Grund auf neu erstellen“.
4. Geben Sie im Abschnitt Basic information (Basisinformationen) folgende Werte ein:
 - a. Geben Sie für Function name (Funktionsname) **TestLRA** ein.
 - b. Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - c. Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.
5. Wählen Sie Funktion erstellen.

Basic information

Function name
Enter a name that describes the purpose of your function.

TestLRA

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

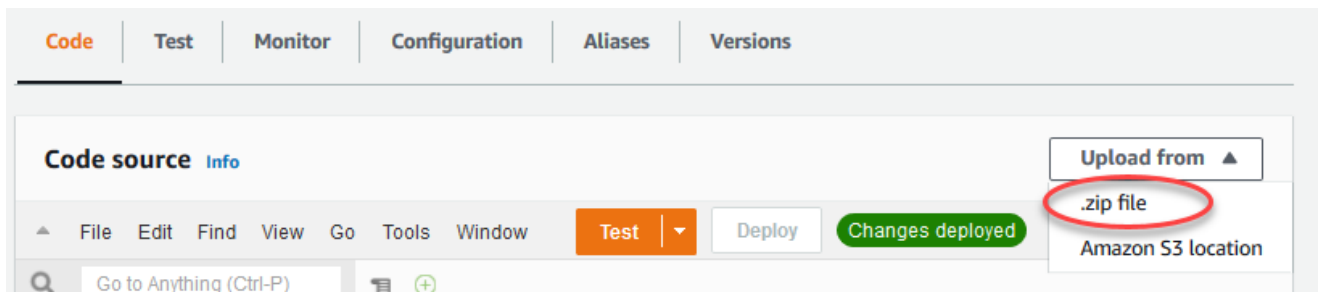
Python 3.7

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

► [Choose or create an execution role](#)

Cancel **Create function**

6. Laden Sie Ihr Lambda-Funktionsbereitstellungspaket hoch und registrieren Sie den Handler.
 - a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Hochladen und dann Ihr `lraTestLambda.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Geben Sie unter Handler die Einstellung `lraTest.function_handler` ein.
- d. Wählen Sie Speichern.

Note

Die Testtaste auf der AWS Lambda Konsole funktioniert mit dieser Funktion nicht. Das AWS IoT Greengrass Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig in der AWS Lambda Konsole auszuführen. Diese Module (zum Beispiel `greengrass_common`) werden den Funktionen zur Verfügung gestellt, nachdem sie auf Ihrem Greengrass-Kern bereitgestellt wurden.

Veröffentlichen Sie als Nächstes die erste Version Ihrer Lambda-Funktion. Anschließend erstellen Sie einen [Alias für die Version](#).

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppengröße nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

7. Wählen Sie im Menü Actions (Aktionen) die Option Publish new version (Neue Version veröffentlichen) aus.

8. Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.
9. Wählen Sie auf der Konfigurationsseite für TestLRA: 1 unter Actions (Aktionen) die Option Create alias (Alias erstellen) aus.
10. Geben Sie auf der Seite Alias erstellen als Namen den Wert **test**. Geben Sie für Version 1 ein.

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

11. Wählen Sie Erstellen aus.

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name*

test

Description

Version*

1

You can shift traffic between two versions, based on weights (%) that you assign. Click [here](#) to learn more.

Additional version

Cancel

Create

Sie können jetzt die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzufügen.

Schritt 3: Fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu

In diesem Schritt fügen Sie die Funktion Ihrer Gruppe hinzu und konfigurieren dann den Lebenszyklus der Funktion.

Fügen Sie zunächst die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzu.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Greengrass-Gruppe aus, der Sie die Lambda-Funktion hinzufügen möchten.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
4. Wählen Sie im Abschnitt Meine Lambda-Funktionen die Option Hinzufügen aus.
5. Wählen Sie auf der Seite Lambda-Funktion hinzufügen die Lambda-Funktion aus. Wählen Sie **TestLRA**.
6. Wählen Sie die Lambda-Funktionsversion.
7. Wählen Sie im Abschnitt Lambda-Funktionskonfiguration die Optionen Systembenutzer und -gruppe und Lambda-Funktionscontainerisierung aus.

Als Nächstes konfigurieren Sie den Lebenszyklus der Lambda-Funktion.

8. Wählen Sie für Timeout 30 seconds (30 Sekunden).

 Important

Lambda-Funktionen, die lokale Ressourcen verwenden (wie in diesem Verfahren beschrieben), müssen in einem Greengrass-Container ausgeführt werden. Andernfalls wird die Bereitstellung der Funktion fehlschlagen. Weitere Informationen finden Sie unter [Containerization \(Containerisierung\)](#).

9. Wählen Sie unten auf der Seite Lambda-Funktion hinzufügen aus.

Schritt 4: Hinzufügen einer lokalen Ressource zur Greengrass-Gruppe

In diesem Schritt fügen Sie eine lokale Volume-Ressource der Greengrass-Gruppe hinzu und erteilen der Funktion Lese- und Schreibzugriff auf die Ressource. Eine lokale Ressource verfügt über einen Gruppenebenen-Bereich. Sie können jeder Lambda-Funktion in der Gruppe Berechtigungen für den Zugriff auf die Ressource gewähren.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen aus.
2. Wählen Sie im Abschnitt Lokale Ressourcen die Option Hinzufügen aus.
3. Verwenden Sie auf der Seite Lokale Ressource hinzufügen die folgenden Werte.

- a. Geben Sie für Resource name (Ressourcenname) **testDirectory** ein.
- b. Wählen Sie für Resource type (Ressourcentyp) die Option Volume aus.
- c. Geben Sie für Lokaler Gerätepfad den Wert ein **/src/LRAtest**. Dieser Pfad muss auf dem Host-Betriebssystem vorhanden sein.

Der lokale Gerätepfad ist der lokale absolute Pfad der Ressource im Dateisystem des Kerngeräts. Dieser Speicherort befindet sich außerhalb des [Containers](#), indem die Funktion ausgeführt wird. Der Pfad darf nicht mit /sys beginnen.

- d. Geben Sie für Destination path (Zielpfad) **/dest/LRAtest** ein. Dieser Pfad muss auf dem Host-Betriebssystem vorhanden sein.

Der Zielpfad ist der absolute Pfad der Ressource im Lambda-Namespace. Dieser Speicherort befindet sich innerhalb des Containers, indem die Funktion ausgeführt wird.

- e. Wählen Sie unter Besitzer der Systemgruppe und Dateizugriffsberechtigung die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.

Mit der Option Systemgruppenbesitzer und Dateizugriffsberechtigung können Sie dem Lambda-Prozess zusätzliche Dateizugriffsberechtigungen gewähren. Weitere Informationen finden Sie unter [Dateizugriffsberechtigung des Gruppenbesitzers](#).

4. Wählen Sie Add resource (Ressource hinzufügen) aus. Auf der Seite Resources wird die neue Ressource "testDirectory" angezeigt.

Schritt 5: Hinzufügen von Abonnements zur Greengrass-Gruppe

In diesem Schritt fügen Sie zwei Abonnements zur Greengrass-Gruppe hinzu. Diese Abonnements ermöglichen die bidirektionale Kommunikation zwischen der Lambda-Funktion und AWS IoT.

Erstellen Sie zunächst ein Abonnement für die Lambda-Funktion, an die Nachrichten gesendet werden sollen AWS IoT.

1. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Registerkarte Abonnements aus.
2. Wählen Sie Hinzufügen aus.
3. Konfigurieren Sie auf der Seite Abonnement erstellen die Quelle und das Ziel wie folgt:
 - a. Wählen Sie als Quelltyp die Option Lambda-Funktion und dann TestLRA aus.

- b. Wählen Sie als Zieltyp Service und dann IoT Cloud aus.
 - c. Geben Sie **LRA/test** als Themenfilter Folgendes ein und wählen Sie dann Abonnement erstellen aus.
4. Auf der Seite Subscriptions wird das neue Abonnement angezeigt.

Als Nächstes konfigurieren Sie ein Abonnement, das die Funktion von AWS IoT aufruft.

5. Wählen Sie auf der Seite Subscriptions die Option Add Subscription aus.
6. Konfigurieren Sie auf der Seite Select your source and target die Quelle und das Ziel wie folgt:
 - a. Wählen Sie als Quelltyp Lambda-Funktion und dann IoT Cloud aus.
 - b. Wählen Sie als Zieltyp Service und dann TestLRA aus.
 - c. Wählen Sie Weiter aus.
7. Geben Sie auf der Seite Filter your data with a topic (Filtern Sie Ihre Daten nach einem Thema.) für Topic filter (Themenfilter) **invoke/LRAFunction** ein und wählen Sie dann Next (Weiter).
8. Wählen Sie Finish (Abschließen). Auf der Seite Subscriptions werden die beiden Abonnements angezeigt.

Schritt 6: Stellen Sie die Gruppe bereit AWS IoT Greengrass

In diesem Schritt stellen Sie die aktuelle Version der Gruppendefinition bereit.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/1.11.6/bin/daemon` enthält, dann wird der Daemon ausgeführt.

 Note

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Deploy aus.

 Note

Die Bereitstellung schlägt fehl, wenn Sie Ihre Lambda-Funktion ohne Containerisierung ausführen und versuchen, auf angehängte lokale Ressourcen zuzugreifen.

3. Wenn Sie dazu aufgefordert werden, wählen Sie auf der Registerkarte Lambda-Funktion unter System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten und dann Automatisch erkennen aus.

Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Eine automatische Erkennung wird empfohlen, unterstützt aber AWS IoT Greengrass auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

 Note

Wenn Sie dazu aufgefordert werden, erteilen Sie die Erlaubnis, die [Greengrass-Servicerolle zu erstellen und sie Ihrer AWS-Konto aktuellen AWS-Region Rolle](#) zuzuordnen. Diese Rolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Diensten.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss lautet der Bereitstellungsstatus Abgeschlossen.

Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Testen des lokalen Ressourcenzugriffs

Nun können Sie prüfen, ob der Zugriff auf die lokale Ressource korrekt konfiguriert ist. Um zu testen, abonnieren Sie das Thema `LRA/test` und veröffentlichen Sie im `invoke/LRAFunction`-Thema. Der Test ist erfolgreich, wenn die Lambda-Funktion die erwartete Nutzlast an sendet. AWS IoT

1. Wählen Sie im Navigationsmenü der AWS IoT Konsole unter Test die Option MQTT-Testclient aus.
2. Geben Sie **LRA/test** unter Thema abonnieren für Themenfilter den Wert ein.
3. Wählen Sie unter Zusätzliche Informationen für die Anzeige von MQTT-Payloads die Option Payloads als Zeichenketten anzeigen aus.
4. Wählen Sie Subscribe (Abonnieren) aus. Ihre Lambda-Funktion veröffentlicht zum LRA/test Thema.

Subscribe to a topic | **Publish to a topic**

Topic filter [Info](#)
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

▼ **Additional configuration**

Number of messages to keep
The MQTT test client keeps this many of the most recent messages published to a topic that matches this topic filter.

Quality of service
When subscribing to a topic, quality of service 0 will be chosen by default.

☒ Quality of Service 0 - Message will be delivered at most once
☐ Quality of Service 1 - Message will be delivered at least once

MQTT payload display

☐ Auto-format JSON payloads (improves readability)
☒ Display payloads as strings (more accurate)
☐ Display raw payloads (displays binary data as hexadecimal values)

Subscribe

5. Geben Sie unter In einem Thema veröffentlichen den Namen des Themas ein **invoke/LRAFunction**, und wählen Sie dann Veröffentlichen aus, um Ihre Lambda-Funktion aufzurufen. Der Test ist erfolgreich, wenn auf der Seite die drei Nachrichtennutzlasten der Funktion angezeigt werden.

Subscribe to a topic

Publish to a topic

Topic name

The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

Q invoke/LRAFunction

X

Message payload

```
{  
  "message": "Hello from AWS IoT console"  
}
```

► Additional configuration

Publish

Subscriptions

lra/test

▼

X

lra/test

▼

May 03, 2021, 12:09:18 (UTC-0400)

Successfully write to a file.

lra/test

▼

May 03, 2021, 12:09:06 (UTC-0400)

```
posix.stat_result(st_mode=16893, st_ino=171142L, st_dev=45831L, st_nlink=2, st_uid=0, st  
_gid=119, st_size=4096L, st_atime=1620054520, st_mtime=1620058120, st_ctime=1620058120)
```

lra/test

▼

May 03, 2021, 12:09:04 (UTC-0400)

Sent from Greengrass Core.

Pause

Clear

Export

Edit

Die von der Lambda-Funktion erstellte Testdatei befindet sich im `/src/LRAtest` Verzeichnis auf dem Greengrass-Core-Gerät. Obwohl die Lambda-Funktion in eine Datei im `/dest/LRAtest` Verzeichnis schreibt, ist diese Datei nur im Lambda-Namespace sichtbar. Sie können sie in einem regulären Linux-Namespace nicht sehen. Alle Änderungen am Zielpfad werden im Quellpfad des Dateisystems reflektiert.

Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Durchführen von Machine Learning-Inferenzen

Diese Funktion ist für AWS IoT Greengrass Core v1.6 oder höher verfügbar.

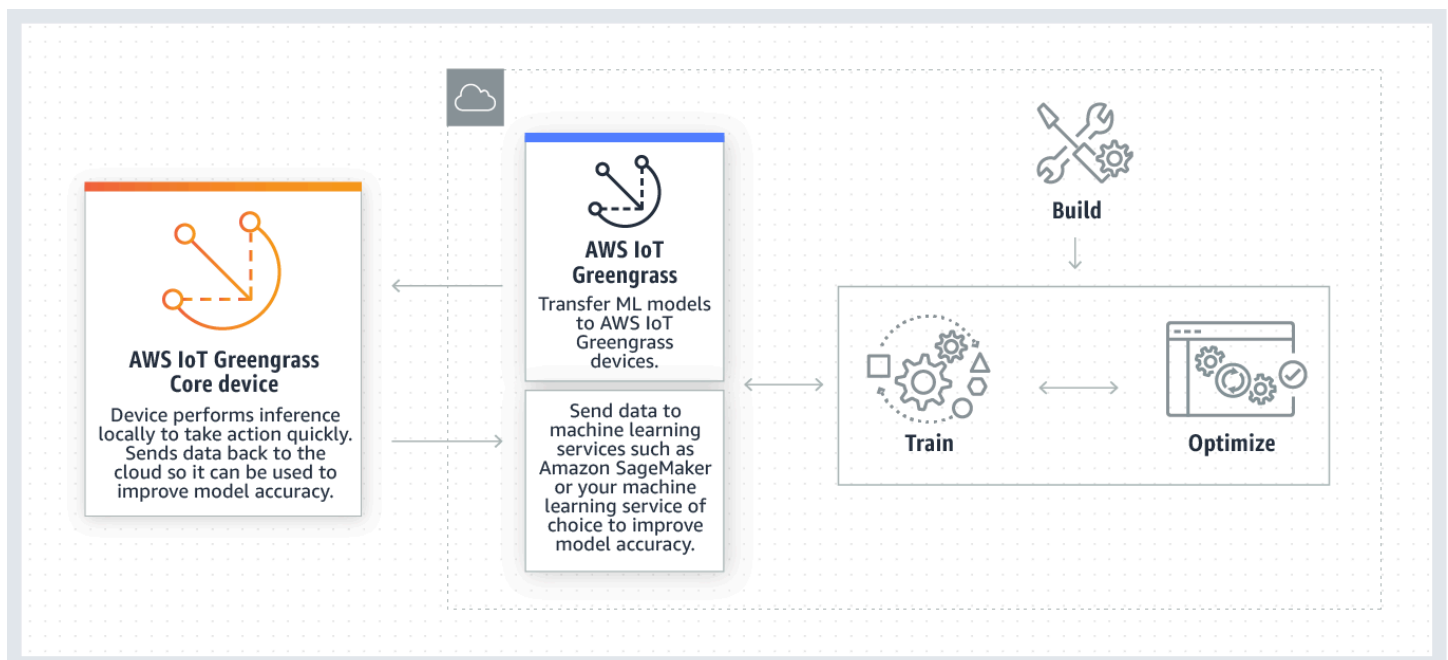
Mit AWS IoT Greengrass können Sie maschinelles Lernen (ML) direkt am Netzwerkrand anhand lokal generierter Daten mithilfe von in der Cloud trainierten Modellen ableiten. Sie können von der niedrigen Latenz und Kosteneinsparungen der Ausführung von lokaler Inferenz profitieren und trotzdem die Cloud-Rechenleistung für Schulungsmodelle und komplexe Verarbeitung nutzen.

Hinweise zur Durchführung von lokaler Inferenz finden Sie unter [the section called “So konfigurieren Sie Machine Learning-Inferenz”](#).

Wie funktioniert AWS IoT Greengrass ML-Inferenz

Sie können Ihre Inferenzmodelle überall trainieren, sie lokal als Ressourcen für maschinelles Lernen in einer Greengrass-Gruppe bereitstellen und dann über Greengrass Lambda-Funktionen auf sie zugreifen. Sie können beispielsweise Deep-Learning-Modelle in [SageMaker KI](#) erstellen und trainieren und sie auf Ihrem Greengrass-Kern einsetzen. Anschließend können Ihre Lambda-Funktionen die lokalen Modelle verwenden, um Inferenzen auf verbundenen Geräten durchzuführen und neue Trainingsdaten zurück in die Cloud zu senden.

Das folgende Diagramm zeigt den AWS IoT Greengrass ML-Inferenz-Workflow.



AWS IoT Greengrass ML-Inferenz vereinfacht jeden Schritt des ML-Workflows, einschließlich:

- Die Entwicklung und Bereitstellung von ML-Framework-Prototypen.
- Der Zugriff auf Cloud-geschulte Modelle und die deren Bereitstellung in Greengrass Core-Geräten.
- [Erstellung von Inferenz-Apps, die auf Hardwarebeschleuniger \(wie GPUs und FPGAs\) als lokale Ressourcen zugreifen können.](#)

Machine Learning-Ressourcen

Ressourcen für maschinelles Lernen stellen in der Cloud trainierte Inferenzmodelle dar, die für einen Kern bereitgestellt werden. AWS IoT Greengrass Um Ressourcen für maschinelles Lernen bereitzustellen, fügen Sie die Ressourcen zunächst einer Greengrass-Gruppe hinzu und definieren dann, wie Lambda-Funktionen in der Gruppe auf sie zugreifen können. AWS IoT Greengrass Ruft während der Gruppenbereitstellung die Quellmodellpakete aus der Cloud ab und extrahiert sie in Verzeichnisse innerhalb des Lambda-Laufzeit-Namespace. Anschließend verwenden die Lambda-Funktionen von Greengrass die lokal bereitgestellten Modelle, um Inferenzen durchzuführen.

Um ein lokal bereitgestelltes Modell zu aktualisieren, aktualisieren Sie zuerst das Quellmodell (in der Cloud), das mit der Machine Learning-Ressource übereinstimmt, und stellen Sie dann die Gruppe bereit. Während der Bereitstellung überprüft AWS IoT Greengrass die Quelle auf Änderungen. Wenn Änderungen erkannt werden, wird das lokale AWS IoT Greengrass Modell aktualisiert.

Unterstützte Modellquellen

AWS IoT Greengrass unterstützt SageMaker KI- und Amazon S3 S3-Modellquellen für Ressourcen für maschinelles Lernen.

Die folgenden Anforderungen gelten für Modellquellen:

- S3-Buckets, in denen Ihre SageMaker KI- und Amazon S3 S3-Modellquellen gespeichert sind, dürfen nicht mit SSE-C verschlüsselt werden. Für Buckets, die serverseitige Verschlüsselung verwenden, unterstützt AWS IoT Greengrass ML-Inferenz derzeit nur die Verschlüsselungsoptionen SSE-S3 oder SSE-KMS. Weitere Informationen zu serverseitigen Verschlüsselungsoptionen finden Sie unter [Schützen von Daten mithilfe serverseitiger Verschlüsselung](#) im Amazon Simple Storage Service-Benutzerhandbuch.
- Die Namen der S3-Buckets, in denen Ihre SageMaker KI- und Amazon S3 S3-Modellquellen gespeichert sind, dürfen keine Punkte (.) enthalten. Weitere Informationen finden Sie in der Regel zur Verwendung von Buckets im virtuellen Hosted-Stil mit SSL unter [Regeln für die Bucket-Benennung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

- [Service AWS-Region Level-Support](#) muss sowohl für KI als auch für KI verfügbar sein. [AWS IoT GreengrassSageMaker](#) AWS IoT Greengrass Unterstützt derzeit SageMaker KI-Modelle in den folgenden Regionen:
 - US East (Ohio)
 - USA Ost (Nord-Virginia)
 - USA West (Oregon)
 - Asia Pacific (Mumbai)
 - Asia Pacific (Seoul)
 - Asien-Pazifik (Singapur)
 - Asien-Pazifik (Sydney)
 - Asien-Pazifik (Tokio)
 - Europe (Frankfurt)
 - Europa (Irland)
 - Europa (London)
- AWS IoT Greengrass muss über read Berechtigungen für die Modellquelle verfügen, wie in den folgenden Abschnitten beschrieben.

SageMaker AI

AWS IoT Greengrass unterstützt Modelle, die als SageMaker KI-Trainingsjobs gespeichert wurden. SageMaker KI ist ein vollständig verwalteter ML-Service, mit dem Sie Modelle mithilfe integrierter oder benutzerdefinierter Algorithmen erstellen und trainieren können. Weitere Informationen finden Sie unter [Was ist SageMaker KI?](#) im SageMaker AI Developer Guide.

Wenn Sie Ihre SageMaker KI-Umgebung konfiguriert haben, indem Sie [einen Bucket erstellt](#) haben, dessen Name enthält `sagemaker`, dann AWS IoT Greengrass über ausreichende Berechtigungen für den Zugriff auf Ihre SageMaker KI-Schulungsjobs verfügt. Die verwaltete Richtlinie `AWSGreengrassResourceAccessRolePolicy` erlaubt den Zugriff auf Buckets, deren Name die Zeichenfolge `sagemaker` enthält. Diese Richtlinie ist der [Greengrass-Servicerolle](#) angefügt.

Andernfalls müssen Sie dem Bucket, in dem Ihr Trainingsjob gespeichert ist, die AWS IoT Greengrass read Erlaubnis erteilen. Betten Sie zu diesem Zweck die folgende eingebundene Richtlinie in die Greengrass-Servicerolle ein. Sie können mehrere Buckets auflisten ARNs.

JSON

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

Amazon S3

AWS IoT Greengrass unterstützt Modelle, die in Amazon S3 als tar.gz .zip OR-Dateien gespeichert sind.

Um auf Modelle zugreifen AWS IoT Greengrass zu können, die in Amazon S3 S3-Buckets gespeichert sind, müssen Sie die AWS IoT Greengrass read Erlaubnis zum Zugriff auf die Buckets erteilen, indem Sie einen der folgenden Schritte ausführen:

- Speichern Sie Ihr Modell in einem Bucket, dessen Name greengrass enthält.

Die verwaltete Richtlinie `AWSGreengrassResourceAccessRolePolicy` erlaubt den Zugriff auf Buckets, deren Name die Zeichenfolge greengrass enthält. Diese Richtlinie ist der [Greengrass-Servicerolle](#) angefügt.

- Betten Sie eine eingebundene Richtlinie in die Greengrass-Service-Rolle ein.

Wenn Ihr Bucket-Name nicht greengrass enthält, fügen Sie die folgende eingebundene Richtlinien der Service-Rolle hinzu. Sie können mehrere Buckets auflisten. ARNs

JSON

```
{
  "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::amzn-s3-demo-bucket"
    ]
  }
]
```

Weitere Informationen finden Sie unter [Einbetten von Inline-Richtlinien](#) im IAM-Benutzerhandbuch.

Voraussetzungen

Die folgenden Anforderungen gelten für die Erstellung und Verwendung von Machine Learning-Ressourcen:

- Sie müssen AWS IoT Greengrass Core v1.6 oder höher verwenden.
- Benutzerdefinierte Lambda-Funktionen können `read` und `write` OR-Operationen auf der `read` Ressource ausführen. Berechtigungen für andere Operationen sind nicht verfügbar. Der Containerisierungsmodus verbundener Lambda-Funktionen bestimmt, wie Sie Zugriffsberechtigungen festlegen. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Machine Learning-Ressourcen”](#).
- Sie müssen den vollständigen Pfad der Ressource im Betriebssystem des Core-Geräts bereitstellen.
- Der Name oder die ID einer Ressource kann maximal 128 Zeichen umfassen und muss dem Muster `[a-zA-Z0-9:_-]+` entsprechen.

Laufzeiten und Bibliotheken für ML-Inferenz

Sie können die folgenden ML-Laufzeiten und -Bibliotheken mit verwenden. AWS IoT Greengrass

- [Deep-Learning-Laufzeit von Amazon SageMaker Neo](#)

- Apache MXNet
- TensorFlow

Diese Laufzeiten und Bibliotheken können auf den Plattformen NVIDIA Jetson TX2, Intel Atom und Raspberry Pi installiert werden. Informationen zum Download finden Sie unter [the section called “Unterstützte Machine Learning-Laufzeiten und -Bibliotheken”](#). Sie können sie direkt auf Ihrem Core-Gerät installieren.

Lesen Sie die folgenden Informationen zu Kompatibilität und Einschränkungen.

SageMaker Deep-Learning-Laufzeit von AI Neo

Sie können die SageMaker AI Neo Deep Learning-Laufzeit verwenden, um Inferenzen mit optimierten Modellen für maschinelles Lernen auf Ihren AWS IoT Greengrass Geräten durchzuführen. Diese Modelle werden mithilfe des SageMaker AI Neo Deep-Learning-Compilers optimiert, um die Geschwindigkeit der Vorhersage von Inferenzen beim maschinellen Lernen zu verbessern. Weitere Informationen zur Modelloptimierung in SageMaker KI finden Sie in der [SageMaker AI Neo-Dokumentation](#).

Note

Derzeit können Sie Modelle für maschinelles Lernen mit dem Neo Deep Learning-Compiler nur in bestimmten Amazon Web Services Services-Regionen optimieren. Sie können jedoch die Neo-Deep-Learning-Laufzeit mit optimierten Modellen in allen Modellen verwenden, in AWS-Region denen der AWS IoT Greengrass Kern unterstützt wird. Weitere Informationen finden Sie unter [So konfigurieren Sie Optimized Machine Learning-Inferenz](#).

MXNet Versionierung

Apache gewährleistet derzeit MXNet keine Aufwärtskompatibilität, sodass Modelle, die Sie mit späteren Versionen des Frameworks trainieren, in früheren Versionen des Frameworks möglicherweise nicht richtig funktionieren. Verwenden Sie in beiden Phasen dieselbe MXNet Framework-Version, um Konflikte zwischen den Phasen Modelltraining und Modellbereitstellung zu vermeiden und ein einheitliches end-to-end Benutzererlebnis zu gewährleisten.

MXNet auf Raspberry Pi

Greengrass Lambda-Funktionen, die auf lokale MXNet Modelle zugreifen, müssen die folgende Umgebungsvariable setzen:

```
MXNET_ENGINE_TYPE=NativeEngine
```

Sie können die Umgebungsvariable im Funktionscode festlegen oder zur gruppenspezifischen Konfiguration der Funktion hinzufügen. Ein Beispiel, bei dem dieses im Rahmen der Konfigurationseinstellung hinzugefügt wird, finden Sie in diesem [Schritt](#).

Note

Für die allgemeine Verwendung des MXNet Frameworks, z. B. die Ausführung eines Codebeispiels eines Drittanbieters, muss die Umgebungsvariable auf dem Raspberry Pi konfiguriert werden.

TensorFlow Einschränkungen bei der Modellverwaltung auf dem Raspberry Pi

Die folgenden Empfehlungen zur Verbesserung der Inferenzergebnisse basieren auf unseren Tests mit den TensorFlow 32-Bit-ARM-Bibliotheken auf der Raspberry Pi-Plattform. Diese Empfehlungen sind für erfahrene Benutzer nur zur Referenz ohne Gewährleistung jedweder Art bestimmt.

- Modelle, die mit dem [Checkpoint](#)-Format geschult werden, sollten vor der Bereitstellung im Protokollpufferformat "eingefroren" werden. Ein Beispiel finden Sie in der [Modellbibliothek zur Bildklassifizierung TensorFlow -Slim](#).
- Verwenden Sie die TF-Estimator- und TF-Slim-Bibliotheken nicht in Schulungen oder Inferenzcodes. Stattdessen verwenden Sie das Modellbereitstellungsmuster der .pb-Datei wie im folgenden Beispiel gezeigt.

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
    tf.import_graph_def(graph_def)
```

 Note

Weitere Informationen zu den unterstützten Plattformen für TensorFlow finden Sie TensorFlow in der TensorFlow Dokumentation unter [Installation](#).

Greifen Sie über Lambda-Funktionen auf Ressourcen für maschinelles Lernen zu

Benutzerdefinierte Lambda-Funktionen können auf Ressourcen für maschinelles Lernen zugreifen, um lokale Inferenzen auf dem Kern auszuführen. AWS IoT Greengrass Eine Machine Learning-Ressource besteht aus dem trainierten Modell und anderen Artefakten, die auf das Core-Gerät heruntergeladen werden.

Damit eine Lambda-Funktion auf eine maschinelle Lernressource auf dem Core zugreifen kann, müssen Sie die Ressource an die Lambda-Funktion anhängen und Zugriffsberechtigungen definieren. Der [Containerisierungsmodus](#) der verbundenen (oder angehängten) Lambda-Funktion bestimmt, wie Sie dies tun.

Zugriffsberechtigungen für Machine Learning-Ressourcen

Ab AWS IoT Greengrass Core v1.10.0 können Sie einen Ressourcenbesitzer für eine maschinelle Lernressource definieren. Der Ressourcenbesitzer steht für die Betriebssystemgruppe und die Berechtigungen, die zum Herunterladen der Ressourcenartefakte AWS IoT Greengrass verwendet werden. Wenn ein Ressourcenbesitzer nicht definiert ist, sind die heruntergeladenen Ressourcenartefakte nur für Root zugänglich.

- Wenn Lambda-Funktionen, die nicht containerisiert sind, auf eine Ressource für maschinelles Lernen zugreifen, müssen Sie einen Ressourcenbesitzer definieren, da es keine Berechtigungssteuerung durch den Container gibt. Lambda-Funktionen, die nicht containerisiert sind, können die Berechtigungen des Ressourcenbesitzers erben und sie für den Zugriff auf die Ressource verwenden.
- Wenn nur containerisierte Lambda-Funktionen auf die Ressource zugreifen, empfehlen wir, dass Sie Berechtigungen auf Funktionsebene verwenden, anstatt einen Ressourcenbesitzer zu definieren.

Eigenschaften der Ressourcenbesitzer

Ein Ressourcenbesitzer gibt einen Gruppenbesitzer und Gruppenbesitzer-Berechtigungen an.

Gruppenbesitzer. Die ID der Gruppe (GID) einer vorhandenen Linux-Betriebssystemgruppe auf dem Core-Gerät. Die Berechtigungen der Gruppe werden dem Lambda-Prozess hinzugefügt. Insbesondere wird die GID zur ergänzenden Gruppe IDs der Lambda-Funktion hinzugefügt.

Wenn eine Lambda-Funktion in der Greengrass-Gruppe so konfiguriert ist, dass sie in derselben Betriebssystemgruppe [wie der Ressourcenbesitzer für eine maschinelle Lernressource ausgeführt](#) wird, muss die Ressource an die Lambda-Funktion angehängt werden. Andernfalls schlägt die Bereitstellung fehl, da diese Konfiguration implizite Berechtigungen gewährt, mit denen die Lambda-Funktion ohne AWS IoT Greengrass Autorisierung auf die Ressource zugreifen kann. Die Überprüfung der Bereitstellungsvalidierung wird übersprungen, wenn die Lambda-Funktion als Root (UID=0) ausgeführt wird.

Wir empfehlen, dass Sie eine Betriebssystemgruppe verwenden, die nicht von anderen Ressourcen, Lambda-Funktionen oder Dateien auf dem Greengrass-Kern verwendet wird. Durch die Verwendung einer gemeinsam genutzten Betriebssystemgruppe erhalten angehängte Lambda-Funktionen mehr Zugriffsberechtigungen, als sie benötigen. Wenn Sie eine gemeinsam genutzte Betriebssystemgruppe verwenden, muss eine angefügte Lambda-Funktion auch an alle Ressourcen für maschinelles Lernen angehängt werden, die die gemeinsam genutzte Betriebssystemgruppe verwenden. Andernfalls schlägt die Bereitstellung fehl.

Gruppenbesitzer-Berechtigungen. Die Nur-Lese- oder Lese- und Schreibberechtigung, die zum Lambda-Prozess hinzugefügt werden soll.

Nicht containerisierte Lambda-Funktionen müssen diese Zugriffsberechtigungen für die Ressource erben. Containerisierte Lambda-Funktionen können diese Berechtigungen auf Ressourcenebene erben oder Berechtigungen auf Funktionsebene definieren. Wenn sie Berechtigungen auf Funktionsebene definieren, müssen die Berechtigungen dieselben oder restriktiver sein als die Berechtigungen auf Ressourcenebene.

Die folgende Tabelle zeigt unterstützte Zugriffsberechtigungskonfigurationen.

GGC v1.10 or later

Eigenschaft	Wenn nur containerisierte Lambda-Funktionen auf die Ressource zugreifen	Wenn nicht containerisierte Lambda-Funktionen auf die Ressource zugreifen
Eigenschaften auf Funktionsebene		
Berechtigungen (Lesen/Schreiben)	Erforderlich, es sei denn, die Ressource definiert einen Ressourcenbesitzer . Wenn ein Ressourcenbesitzer definiert ist, müssen Berechtigungen auf Funktionsebene dieselben oder restriktiver sein als die Ressourcenbesitzer-Berechtigungen. Wenn nur containerisierte Lambda-Funktionen auf die Ressource zugreifen , empfehlen wir, keinen Ressourcenbesitzer zu definieren.	Nicht containerisierte Lambda-Funktionen: Nicht unterstützt Lambda-Funktionen, die nicht containerisiert sind, müssen Berechtigungen auf Ressourcenebene erben. Containerisierte Lambda-Funktionen: Optional, müssen jedoch dieselben oder restriktiver sein als Berechtigungen auf Ressourcenebene.
Eigenschaften auf Ressourcenebene		
Ressourcenbesitzer	Optional (nicht empfohlen).	Erforderlich
Berechtigungen (Lesen/Schreiben)	Optional (nicht empfohlen).	Erforderlich

GGC v1.9 or earlier

Eigenschaft	Wenn nur containerisierte Lambda-Funktionen auf die Ressource zugreifen	Wenn nicht containerisierte Lambda-Funktionen auf die Ressource zugreifen
Eigenschaften auf Funktionsebene		
Berechtigungen (Lesen/Schreiben)	Erforderlich	Nicht unterstützt
Eigenschaften auf Ressourcenebene		
Ressourcenbesitzer	Nicht unterstützt	Nicht unterstützt
Berechtigungen (Lesen/Schreiben)	Nicht unterstützt	Nicht unterstützt

 Note

Wenn Sie die AWS IoT Greengrass API verwenden, um Lambda-Funktionen und -Ressourcen zu konfigurieren, ist auch die `ResourceId` Eigenschaft auf Funktionsebene erforderlich. Die `ResourceId` Eigenschaft ordnet die Ressource für maschinelles Lernen der Lambda-Funktion zu.

Zugriffsberechtigungen für Lambda-Funktionen definieren (Konsole)

In der AWS IoT Konsole definieren Sie Zugriffsberechtigungen, wenn Sie eine Machine Learning-Ressource konfigurieren oder eine Ressource an eine Lambda-Funktion anhängen.

Containerisierte Lambda-Funktionen

Wenn nur containerisierte Lambda-Funktionen an die Ressource für maschinelles Lernen angehängt sind:

- Wählen Sie Keine Systemgruppe als Ressourcenbesitzer für die maschinelle Lernressource aus. Dies ist die empfohlene Einstellung, wenn nur containerisierte Lambda-Funktionen auf die

Ressource für maschinelles Lernen zugreifen. Andernfalls könnten Sie angehängten Lambda-Funktionen mehr Zugriffsberechtigungen erteilen, als sie benötigen.

Lambda-Funktionen ohne Container (erfordert GGC v1.10 oder höher)

Wenn Lambda-Funktionen, die nicht containerisiert sind, an die Ressource für maschinelles Lernen angehängt sind:

- Geben Sie die Systemgruppen-ID (GID) an, die als Ressourcenbesitzer für die maschinelle Lernressource verwendet werden soll. Wählen Sie Systemgruppe und Berechtigungen angeben und geben Sie die GID ein. Sie können den `getent group` Befehl auf Ihrem Core-Gerät verwenden, um die ID einer Systemgruppe nachzuschlagen.
- Wählen Sie Nur-Lese-Zugriff oder Lese- und Schreibzugriff für die Systemgruppenberechtigungen.

Definition von Zugriffsberechtigungen für Lambda-Funktionen (API)

In der AWS IoT Greengrass API definieren Sie Berechtigungen für maschinelles Lernen in der `ResourceAccessPolicy` Eigenschaft für die Lambda-Funktion oder der `OwnerSetting` Eigenschaft für die Ressource.

Containerisierte Lambda-Funktionen

Wenn nur containerisierte Lambda-Funktionen an die Ressource für maschinelles Lernen angehängt sind:

- Definieren Sie für containerisierte Lambda-Funktionen Zugriffsberechtigungen in der `Permission` Eigenschaft der Eigenschaft. `ResourceAccessPolicies` Beispiel:

```
"Functions": [  
  {  
    "Id": "my-containerized-function",  
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-  
name:alias-or-version",  
    "FunctionConfiguration": {  
      "Environment": {
```

```

        "ResourceAccessPolicies": [
            {
                "ResourceId": "my-resource-id",
                "Permission": "ro-or-rw"
            }
        ],
        "MemorySize": 512,
        "Pinned": true,
        "Timeout": 5
    }
}
]

```

- Bei Machine Learning-Ressourcen lassen Sie die `OwnerSetting`-Eigenschaft weg. Beispiel:

```

"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package"
            }
        }
    }
]

```

Dies ist die empfohlene Konfiguration, wenn nur containerisierte Lambda-Funktionen auf die Ressource für maschinelles Lernen zugreifen. Andernfalls könnten Sie angehängten Lambda-Funktionen mehr Zugriffsberechtigungen erteilen, als sie benötigen.

Lambda-Funktionen ohne Container (erfordert GGC v1.10 oder höher)

Wenn Lambda-Funktionen, die nicht containerisiert sind, an die Ressource für maschinelles Lernen angehängt sind:

- Lassen Sie bei Lambda-Funktionen, die nicht containerisiert sind, die Eigenschaft `PermissionResourceAccessPolicies` weg. Diese Konfiguration ist erforderlich und ermöglicht es der Funktion, die Berechtigung auf Ressourcenebene zu erben. Beispiel:

```

"Functions": [
  {
    "Id": "my-non-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "Execution": {
          "IsolationMode": "NoContainer",
        },
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id"
          }
        ]
      },
      "Pinned": true,
      "Timeout": 5
    }
  }
]

```

- Lassen Sie bei containerisierten Lambda-Funktionen, die auch auf die Ressource für maschinelles Lernen zugreifen, die Permission Eigenschaft weg ResourceAccessPolicies oder definieren Sie eine Berechtigung, die mit der Berechtigung auf Ressourcenebene identisch oder restriktiver ist. Beispiel:

```

"Functions": [
  {
    "Id": "my-containerized-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
    "FunctionConfiguration": {
      "Environment": {
        "ResourceAccessPolicies": [
          {
            "ResourceId": "my-resource-id",
            "Permission": "ro-or-rw" // Optional, but cannot exceed
            the GroupPermission defined for the resource.
          }
        ]
      },
    },
  }
]

```

```

        "MemorySize": 512,
        "Pinned": true,
        "Timeout": 5
      }
    }
  ]

```

- Definieren Sie für Machine Learning-Ressourcen die OwnerSetting-Eigenschaft, einschließlich der untergeordneten GroupOwner- und GroupPermission-Eigenschaften. Beispiel:

```

"Resources": [
  {
    "Id": "my-resource-id",
    "Name": "my-resource-name",
    "ResourceDataContainer": {
      "S3MachineLearningModelResourceData": {
        "DestinationPath": "/local-destination-path",
        "S3Uri": "s3://uri-to-resource-package",
        "OwnerSetting": {
          "GroupOwner": "os-group-id",
          "GroupPermission": "ro-or-rw"
        }
      }
    }
  }
]

```

Zugreifen auf Ressourcen für maschinelles Lernen über den Lambda-Funktionscode

Benutzerdefinierte Lambda-Funktionen verwenden plattformspezifische Betriebssystemsschnittstellen, um auf Ressourcen für maschinelles Lernen auf einem Kerngerät zuzugreifen.

GGC v1.10 or later

Bei containerisierten Lambda-Funktionen wird die Ressource im Greengrass-Container bereitgestellt und ist auf dem für die Ressource definierten lokalen Zielpfad verfügbar. Bei nicht containerisierten Lambda-Funktionen wird die Ressource mit einem Lambda-spezifischen

Arbeitsverzeichnis verknüpft und an die Umgebungsvariable im Lambda-Prozess übergeben.

AWS_GG_RESOURCE_PREFIX

Um den Pfad zu den heruntergeladenen Artefakten einer maschinellen Lernressource abzurufen, hängen Lambda-Funktionen die AWS_GG_RESOURCE_PREFIX Umgebungsvariable an den für die Ressource definierten lokalen Zielpfad an. Für containerisierte Lambda-Funktionen ist der zurückgegebene Wert ein einzelner Schrägstrich (/).

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

Die heruntergeladenen Artefakte einer Machine Learning-Ressource befinden sich im lokalen Zielpfad, der für die Ressource definiert ist. Nur containerisierte Lambda-Funktionen können in AWS IoT Greengrass Core v1.9 und früheren Versionen auf Ressourcen für maschinelles Lernen zugreifen.

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

Ihre Modellladeimplementierung hängt von Ihrer ML-Bibliothek ab.

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Zugriff auf Machine Learning-Ressourcen zu beheben.

Themen

- [Ungültiger MLModel Besitzer — GroupOwnerSetting ist in der ML-Modellressource angegeben, aber GroupOwner oder ist nicht vorhanden GroupPermission](#)
- [NoContainer Die Funktion kann beim Anhängen von Ressourcen für Machine Learning keine Berechtigungen konfigurieren. <function-arn>bezieht sich auf eine Ressource für maschinelles Lernen <resource-id>mit der entsprechenden Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.](#)

- [Die Funktion <function-arn>bezieht sich auf Machine Learning Learning-Ressource, für die <resource-id> ResourceAccessPolicy sowohl die Berechtigungen als auch die Ressource fehlen OwnerSetting.](#)
- [Die Funktion <function-arn>bezieht sich auf die Machine Learning Learning-Ressource <resource-id>mit der Berechtigung\ "rw\“, während die Einstellung für den Ressourcenbesitzer GroupPermission nur\ "ro\“ zulässt.](#)
- [NoContainer Die Funktion <function-arn>bezieht sich auf Ressourcen des verschachtelten Zielpfads.](#)
- [Lambda <function-arn> erhält Zugriff auf die Ressource <resource-id>, indem dieselbe Gruppenbesitzer-ID freigegeben wird](#)

Ungültiger MLModel Besitzer — GroupOwnerSetting ist in der ML-Modellressource angegeben, aber GroupOwner oder ist nicht vorhanden GroupPermission

Lösung: Dieser Fehler wird angezeigt, wenn eine maschinelle Lernressource das [ResourceDownloadOwnerSetting](#) Objekt enthält, die erforderliche GroupPermission Eigenschaft GroupOwner oder aber nicht definiert ist. Um dieses Problem zu beheben, definieren Sie die fehlende Eigenschaft.

NoContainer Die Funktion kann beim Anhängen von Ressourcen für Machine Learning keine Berechtigungen konfigurieren. <function-arn>bezieht sich auf eine Ressource für maschinelles Lernen <resource-id>mit der entsprechenden Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.

Lösung: Sie erhalten diesen Fehler, wenn eine nicht containerisierte Lambda-Funktion Berechtigungen auf Funktionsebene für eine maschinelle Lernressource festlegt. Nicht containerisierte Funktionen müssen Berechtigungen von den Ressourcenbesitzer-Berechtigungen erben, die für die Machine Learning-Ressource definiert sind. Um dieses Problem zu beheben, können Sie die [Berechtigungen des Ressourcenbesitzers \(Konsole\) erben](#) oder [die Berechtigungen aus der Ressourcenzugriffsrichtlinie \(API\) der Lambda-Funktion entfernen](#).

Die Funktion `<function-arn>` bezieht sich auf Machine Learning Learning-Ressource, für die `<resource-id>` ResourceAccessPolicy sowohl die Berechtigungen als auch die Ressource fehlen OwnerSetting.

Lösung: Sie erhalten diesen Fehler, wenn die Berechtigungen für die maschinelle Lernressource nicht für die angehängte Lambda-Funktion oder die Ressource konfiguriert sind. Um dieses Problem zu beheben, konfigurieren Sie Berechtigungen in der [ResourceAccessPolicy](#)Eigenschaft für die Lambda-Funktion oder der [OwnerSetting](#)Eigenschaft für die Ressource.

Die Funktion `<function-arn>` bezieht sich auf die Machine Learning Learning-Ressource `<resource-id>` mit der Berechtigung `"rw"`, während die Einstellung für den Ressourcenbesitzer GroupPermission nur `"ro"` zulässt.

Lösung: Sie erhalten diesen Fehler, wenn die für die angehängte Lambda-Funktion definierten Zugriffsberechtigungen die für die maschinelle Lernressource definierten Berechtigungen des Ressourcenbesitzers überschreiten. Um dieses Problem zu beheben, legen Sie restriktivere Berechtigungen für die Lambda-Funktion oder weniger restriktive Berechtigungen für den Ressourcenbesitzer fest.

NoContainer Die Funktion `<function-arn>` bezieht sich auf Ressourcen des verschachtelten Zielpfads.

Lösung: Sie erhalten diesen Fehler, wenn mehrere maschinelle Lernressourcen, die an eine nicht containerisierte Lambda-Funktion angehängt sind, denselben Zielpfad oder einen verschachtelten Zielpfad verwenden. Um dieses Problem zu beheben, geben Sie separate Zielpfade für die Ressourcen an.

Lambda `<function-arn>` erhält Zugriff auf die Ressource `<resource-id>`, indem dieselbe Gruppenbesitzer-ID freigegeben wird

Lösung: Sie erhalten diesen Fehler, `runtime.log` wenn dieselbe Betriebssystemgruppe als [Run as Identity der Lambda-Funktion und als Ressourcenbesitzer für eine Machine Learning-Ressource](#) angegeben ist, die Ressource jedoch nicht an die Lambda-Funktion angehängt ist. Diese

Konfiguration gewährt der Lambda-Funktion implizite Berechtigungen, mit denen sie ohne AWS IoT Greengrass Autorisierung auf die Ressource zugreifen kann.

Um dieses Problem zu beheben, verwenden Sie eine andere Betriebssystemgruppe für eine der Eigenschaften oder hängen Sie die Ressource für maschinelles Lernen an die Lambda-Funktion an.

Weitere Informationen finden Sie auch unter

- [Durchführen von Machine Learning-Inferenzen](#)
- [the section called “So konfigurieren Sie Machine Learning-Inferenz”](#)
- [the section called “So konfigurieren Sie optimierte Machine Learning-Inferenz”](#)
- [AWS IoT Greengrass Version 1 API Reference](#)

So konfigurieren Sie die Inferenz für maschinelles Lernen mit dem AWS-Managementkonsole

Um die Schritte in diesem Tutorial ausführen zu können, benötigen Sie AWS IoT Greengrass Core v1.10 oder höher.

Sie können Machine Learning (ML)-Inferenz lokal auf einem Greengrass Core-Gerät mit lokal generierten Daten durchführen. Weitere Informationen einschließlich Anforderungen und Einschränkungen finden Sie unter [Durchführen von Machine Learning-Inferenzen](#).

In diesem Tutorial wird beschrieben, wie Sie mithilfe der AWS-Managementkonsole eine Greengrass-Gruppe konfigurieren, um eine Lambda-Inferenz-App auszuführen, die Bilder von einer Kamera lokal erkennt, ohne Daten an die Cloud zu senden. Die Inferenz-App greift auf das Kameramodul auf einem Raspberry Pi zu und führt die Inferenz mithilfe des Open-Source-Modells aus. [SqueezeNet](#)

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Konfigurieren des Raspberry Pi](#)
2. [Installieren Sie das MXNet Framework](#)
3. [Erstellen eines Modellpakets](#)
4. [Eine Lambda-Funktion erstellen und veröffentlichen](#)
5. [Fügen Sie die Lambda-Funktion zur Gruppe hinzu](#)
6. [Hinzufügen von Ressourcen zur Gruppe](#)

7. [Hinzufügen eines Abonnements zur Gruppe](#)
8. [Bereitstellen der Gruppe](#)
9. [Testen der Anwendung](#)

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Raspberry Pi 4 Modell B oder Raspberry Pi 3 Modell B/B+, eingerichtet und konfiguriert für die Verwendung mit AWS IoT Greengrass. Um Ihren Raspberry Pi mit AWS IoT Greengrass einzurichten, führen Sie das [Greengrass Device Setup-Skript](#) aus oder stellen Sie sicher, dass Sie [Modul 1](#) und [Modul 2](#) von [Erste Schritte mit AWS IoT Greengrass](#) abgeschlossen haben.

Note

Der Raspberry Pi benötigt möglicherweise eine [2,5-A-Stromversorgung](#), um die Deep-Learning-Frameworks auszuführen, die normalerweise für die Bildklassifizierung verwendet werden. Ein Netzteil mit einer niedrigeren Nennleistung kann dazu führen, dass das Gerät neu gestartet wird.

- [Raspberry Pi Kameramodul V2 – 8 Megapixel, 1080p](#). Informationen zur Einrichtung der Kamera finden Sie in der Dokumentation zum Raspberry Pi unter [Anschließen der Kamera](#).
- Eine Greengrass-Gruppe und ein Greengrass Core. Informationen zum Erstellen einer Greengrass-Gruppe oder eines Greengrass-Kerns finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#).

Note

Dieses Tutorial verwendet einen Raspberry Pi, AWS IoT Greengrass unterstützt aber auch andere Plattformen wie [Intel Atom](#) und [NVIDIA TX2 Jetson](#). Im Beispiel für Jetson können Sie statische Bilder anstelle von Bildern verwenden TX2, die von einer Kamera gestreamt wurden. Wenn Sie das TX2 Jetson-Beispiel verwenden, müssen Sie möglicherweise Python 3.6 anstelle von Python 3.7 installieren. Informationen zur Konfiguration Ihres Geräts für die Installation der AWS IoT Greengrass Core-Software finden Sie unter [the section called “Einrichten anderer Geräte”](#).

Für Plattformen von Drittanbietern, die AWS IoT Greengrass dies nicht unterstützen, müssen Sie Ihre Lambda-Funktion im Modus ohne Container ausführen. Um im Modus ohne

Container ausgeführt zu werden, müssen Sie Ihre Lambda-Funktion als Root ausführen. Weitere Informationen erhalten Sie unter [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#) und [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

Schritt 1: Konfigurieren des Raspberry Pi

In diesem Schritt installieren Sie Updates für das Raspbian-Betriebssystem, installieren die Kameramodulsoftware und Python-Abhängigkeiten und aktivieren die Kameraschnittstelle.

Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus.

1. Installieren von Updates für Raspbian.

```
sudo apt-get update
sudo apt-get dist-upgrade
```

2. Installieren Sie die `picamera`-Schnittstelle für das Kameramodul und andere Python-Bibliotheken, die für dieses Tutorial erforderlich sind.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Überprüfen Sie die Installation:

- Stellen Sie sicher, dass Ihre Python 3.7-Installation `pip` enthält.

```
python3 -m pip
```

Wenn `pip` nicht installiert ist, laden Sie es von der [pip-Website](#) herunter und führen Sie dann den folgenden Befehl aus.

```
python3 get-pip.py
```

- Stellen Sie sicher, dass Ihre Python-Version 3.7 oder höher ist.

```
python3 --version
```

Wenn in der Ausgabe eine frühere Version aufgelistet ist, führen Sie den folgenden Befehl aus.

```
sudo apt-get install -y python3.7-dev
```

- Stellen Sie sicher, dass Setuptools und Picamera erfolgreich installiert wurden.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'  
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Wenn die Ausgabe keine Fehler enthält, ist die Überprüfung erfolgreich.

Note

Wenn die auf Ihrem Gerät installierte ausführbare Python-Datei `python3.7` ist, verwenden Sie `python3.7` anstelle von `python3` für die Befehle in diesem Tutorial. Stellen Sie sicher, dass Ihre pip-Installation der richtigen `python3.7`- oder `python3`-Version zugeordnet ist, um Abhängigkeitsfehler zu vermeiden.

3. Starten Sie den Raspberry Pi neu.

```
sudo reboot
```

4. Öffnen Sie das Raspberry Pi-Konfigurations-Tool.

```
sudo raspi-config
```

5. Verwenden Sie die Pfeiltasten zum Öffnen von Interfacing Options (Verbindungsoptionen) und aktivieren Sie die Kameraschnittstelle. Wenn Sie dazu aufgefordert werden, lassen Sie den Neustart des Geräts zu.
6. Geben Sie den folgenden Befehl ein, um die Einrichtung der Kamera zu testen.

```
raspistill -v -o test.jpg
```

So werden ein Vorschaufenster im Raspberry Pi geöffnet, ein Bild mit dem Namen `test.jpg` in Ihrem aktuellen Verzeichnis gespeichert und Informationen über die Kamera im Raspberry Pi-Terminal angezeigt.

Schritt 2: Installieren Sie das Framework MXNet

Installieren Sie in diesem Schritt MXNet Bibliotheken auf Ihrem Raspberry Pi.

1. Melden Sie sich per Fernzugriff bei Ihrem Raspberry Pi an.

```
ssh pi@your-device-ip-address
```

2. Öffnen Sie die MXNet Dokumentation, öffnen Sie [Installation MXNet](#) und folgen Sie den Anweisungen zur Installation MXNet auf dem Gerät.

Note

Wir empfehlen, Version 1.5.0 zu installieren und MXNet aus dem Quellcode für dieses Tutorial zu bauen, um Gerätekonflikte zu vermeiden.

3. Überprüfen Sie nach der Installation MXNet die folgende Konfiguration:
 - Stellen Sie sicher, dass das `ggc_user` Systemkonto das MXNet Framework verwenden kann.

```
sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

- Stellen Sie sicher, dass NumPy es installiert ist.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Schritt 3: Erstellen Sie ein MXNet Modellpaket

In diesem Schritt erstellen Sie ein Modellpaket, das ein vortrainiertes MXNet Mustermodell enthält, das auf Amazon Simple Storage Service (Amazon S3) hochgeladen werden soll. AWS IoT Greengrass kann ein Modellpaket von Amazon S3 verwenden, sofern Sie das Format `tar.gz` oder `zip` verwenden.

1. Laden Sie auf Ihrem Computer das MXNet Beispiel für Raspberry Pi von [herunterthe section called "Beispiele für Machine Learning"](#).
2. Entpacken Sie die heruntergeladene `mxnet-py3-armv7l.tar.gz`-Datei.
3. Navigieren Sie zum `squeezenet` Verzeichnis .

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
```

Die `squeezenet.zip`-Datei in diesem Verzeichnis ist Ihr Modellpaket. Es enthält SqueezeNet Open-Source-Modellartefakte für ein Bildklassifizierungsmodell. Später laden Sie dieses Modellpaket auf Amazon S3 hoch.

Schritt 4: Eine Lambda-Funktion erstellen und veröffentlichen

In diesem Schritt erstellen Sie ein Lambda-Funktions-Bereitstellungspaket und eine Lambda-Funktion. Veröffentlichen Sie anschließend eine Funktionsversion und erstellen einen Alias.

Erstellen Sie zunächst das Lambda-Funktionsbereitstellungspaket.

1. Navigieren Sie auf Ihrem Computer zum `examples`-Verzeichnis im Beispieldpaket, das Sie in [the section called “Erstellen eines Modellpakets”](#) entpackt haben.

```
cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
```

Das `examples`-Verzeichnis enthält Funktionscode und Abhängigkeiten.

- `greengrassObjectClassification.py` ist der in diesem Tutorial verwendete Inferenzcode. Sie können diesen Code als Vorlage verwenden, um eine eigene Inferenzfunktion zu erstellen.
- `greengrasssdk` ist Version 1.5.0 des AWS IoT Greengrass Core SDK für Python.

Note

Wenn eine neue Version verfügbar ist, können Sie sie herunterladen und die SDK-Version in Ihrem Bereitstellungspaket aktualisieren. Weitere Informationen finden Sie unter [AWS IoT Greengrass Core SDK for Python](#) on GitHub.

2. Komprimieren Sie den Inhalt des `examples`-Verzeichnisses in eine Datei namens `greengrassObjectClassification.zip`. Dies ist Ihr Bereitstellungspaket.

```
zip -r greengrassObjectClassification.zip .
```

Note

Stellen Sie sicher, dass sich die .py-Dateien und -Abhängigkeiten im Stammverzeichnis befinden.

Als Nächstes erstellen Sie die Lambda-Funktion.

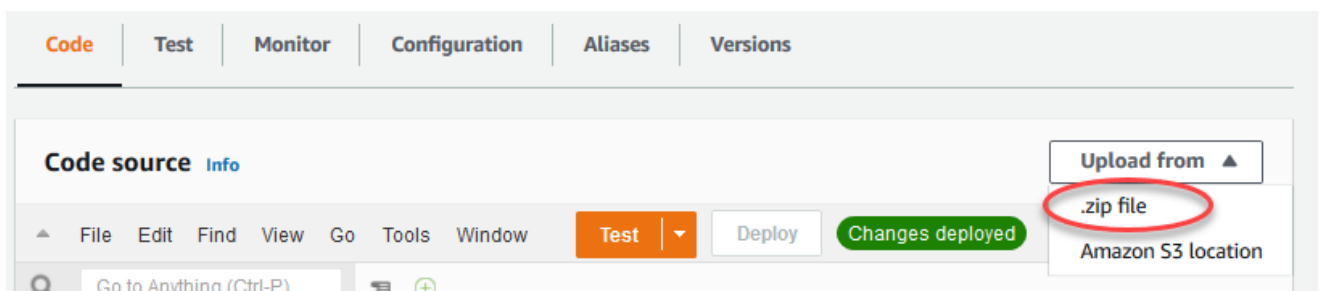
3. Wählen Sie in der AWS IoT Konsole Funktionen und Funktion erstellen aus.
4. Wählen Sie Author von Grund auf neu und verwenden Sie die folgenden Werte, um Ihre Funktion zu erstellen:
 - Geben Sie für Function name (Funktionsname) **greengrassObjectClassification** ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.

5. Wählen Sie Funktion erstellen.

Laden Sie jetzt Ihr Lambda-Funktionsbereitstellungspaket hoch und registrieren Sie den Handler.

6. Wählen Sie Ihre Lambda-Funktion und laden Sie Ihr Lambda-Funktionspaket hoch.
 - a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Hochladen und dann Ihr `greengrassObjectClassification.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Geben Sie unter Handler **`greengrassObjectClassification.function_handler`** ein.

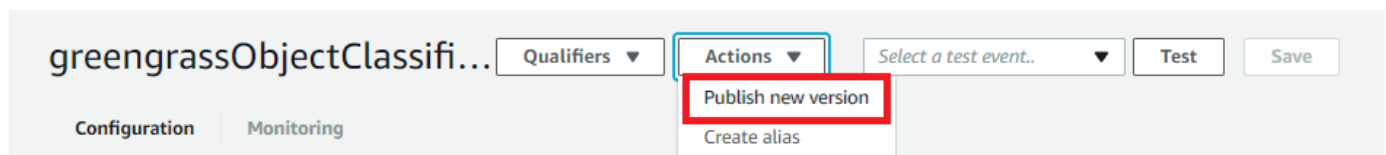
Wählen Sie Speichern.

Veröffentlichen Sie als Nächstes die erste Version Ihrer Lambda-Funktion. Anschließend erstellen Sie einen [Alias für die Version](#).

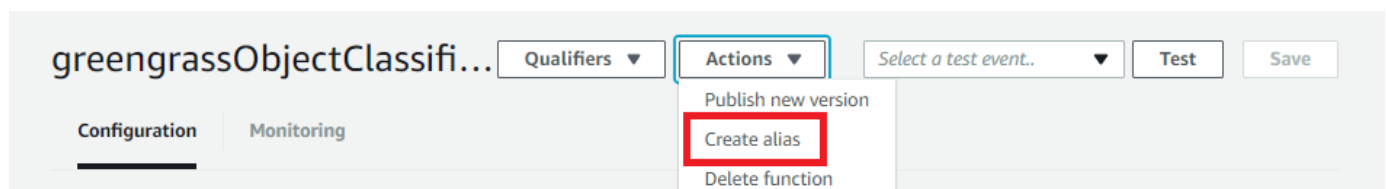
Note

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

7. Wählen Sie im Menü Actions die Option Publish new version aus.



8. Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.
9. Wählen Sie auf der Konfigurationsseite greengrassObjectClassification: 1 im Menü Aktionen die Option Alias erstellen aus.



10. Geben Sie auf der Seite **Create a new alias** folgende Werte an:

- Geben Sie unter **Name** **mlTest** ein.
- Geben Sie in **Version (Version)** **1** ein.

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

11. Wählen Sie **Speichern**.

Fügen Sie nun die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzu.

Schritt 5: Fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu

Fügen Sie in diesem Schritt die Lambda-Funktion zur Gruppe hinzu und konfigurieren Sie dann ihre Lebenszyklus- und Umgebungsvariablen.

Fügen Sie zunächst die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzu.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter **Verwalten** die Option **Greengrass-Geräte** und wählen Sie dann **Gruppen (V1)** aus.
2. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte **Lambda-Funktionen** aus.
3. Wählen Sie im Abschnitt **Meine Lambda-Funktionen** die Option **Hinzufügen** aus.
4. Wählen **greengrassObjectClassification** für die Lambda-Funktion.
5. Wählen Sie für die Lambda-Funktionsversion **alias:MLTest**.

Als Nächstes konfigurieren Sie den Lebenszyklus und die Umgebungsvariablen der Lambda-Funktion.

6. Nehmen Sie im Abschnitt **Lambda-Funktionskonfiguration** die folgenden Aktualisierungen vor.

 Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Systembenutzer und Gruppe die Option. **Another user ID/group ID**. Geben Sie als Systembenutzer-ID ein **0**. Geben Sie als Systemgruppen-ID ein **0**.

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

 Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Kein Container aus.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter. [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Geben Sie in Timeout (Zeitüberschreitung) **10 seconds** ein.
- Wählen Sie für Pinned die Option True aus.

Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

b. Um stattdessen im containerisierten Modus zu starten:

 Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Wählen Sie für Systembenutzer und Gruppe die Option Gruppenstandard verwenden aus.
- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Gruppenstandard verwenden aus.
- Geben Sie in Memory Limit (Speicherlimit) **96 MB** ein.
- Geben Sie in Timeout (Zeitüberschreitung) **10 seconds** ein.
- Wählen Sie für Pinned die Option True aus.

Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

7. Erstellen Sie in Environment variables (Umgebungsvariablen) ein Schlüssel-Wert-Paar. Ein Schlüssel-Wert-Paar ist für Funktionen erforderlich, die mit MXNet Modellen auf einem Raspberry Pi interagieren.

Verwenden Sie als Schlüssel `MXNET_ENGINE_TYPE`. Verwenden Sie für den Wert. `NaiveEngine`

 Note

In Ihren eigenen benutzerdefinierten Lambda-Funktionen können Sie optional die Umgebungsvariable in Ihrem Funktionscode festlegen.

8. Behalten Sie die Standardwerte für alle anderen Eigenschaften bei und wählen Sie Lambda-Funktion hinzufügen.

Schritt 6: Hinzufügen von Ressourcen zur Greengrass-Gruppe

Erstellen Sie in diesem Schritt Ressourcen für das Kameramodul und das ML-Inferenzmodell und ordnen Sie die Ressourcen der Lambda-Funktion zu. Dadurch kann die Lambda-Funktion auf die Ressourcen auf dem Kerngerät zugreifen.

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne diese Geräteressourcen zu konfigurieren.

Erstellen Sie zunächst zwei lokale Geräteressourcen für die Kamera: eine für den gemeinsam genutzten Speicher und eine für die Geräteschnittstelle. Weitere Informationen zum Zugriff auf lokale Ressourcen finden Sie unter [Greifen Sie mit Lambda-Funktionen und Konnektoren auf lokale Ressourcen zu](#).

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen aus.
2. Wählen Sie im Abschnitt Lokale Ressourcen die Option Lokale Ressource hinzufügen aus.
3. Verwenden Sie auf der Seite Lokale Ressource hinzufügen die folgenden Werte:

- Geben Sie für Resource name (Ressourcenname) **videoCoreSharedMemory** ein.
- Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
- Geben Sie für Lokaler Gerätepfad den Wert ein **/dev/vcsm**.

Der Gerätepfad ist der lokale absolute Pfad der Geräteressource. Dieser Pfad kann nur auf ein zeichenorientiertes Gerät oder ein Blockgerät unter `/dev` verweisen.

- Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.

Mit der Option Systemgruppenbesitzer und Dateizugriffsberechtigungen können Sie dem Lambda-Prozess zusätzliche Dateizugriffsberechtigungen gewähren. Weitere Informationen finden Sie unter [Dateizugriffsberechtigung des Gruppenbesitzers](#).

4. Als nächstes fügen Sie eine lokale Geräteressource für die Kameraschnittstelle hinzu.
5. Wählen Sie Lokale Ressource hinzufügen.
6. Verwenden Sie auf der Seite Lokale Ressource hinzufügen die folgenden Werte:
 - Geben Sie für Resource name (Ressourcenname) **videoCoreInterface** ein.
 - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
 - Geben Sie für Lokaler Gerätepfad den Wert ein **/dev/vchiq**.

- Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.
7. Wählen Sie unten auf der Seite die Option Ressource hinzufügen aus.

Fügen Sie nun das Inferenz-Modell als Machine Learning-Ressource hinzu. Dieser Schritt beinhaltet das Hochladen des `squeezenet.zip` Modellpakets auf Amazon S3.

1. Wählen Sie auf der Registerkarte Ressourcen für Ihre Gruppe im Abschnitt Machine Learning die Option Ressource für maschinelles Lernen hinzufügen aus.
2. Geben Sie auf der Seite Ressource für maschinelles Lernen hinzufügen als Ressourcenname den Text `insqueezenet_model1`.
3. Wählen Sie als Modellquelle die Option Ein in S3 gespeichertes Modell verwenden aus, z. B. ein mit dem Deep Learning Compiler optimiertes Modell.
4. Geben Sie für S3-URI einen Pfad ein, in dem der S3-Bucket gespeichert ist.
5. Wählen Sie S3 durchsuchen. Dadurch wird ein neuer Tab zur Amazon S3 S3-Konsole geöffnet.
6. Laden Sie die `squeezenet.zip` Datei auf der Registerkarte Amazon S3 S3-Konsole in einen S3-Bucket hoch. Weitere Informationen finden Sie unter [Wie lade ich Dateien und Ordner in einen S3-Bucket hoch?](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

Damit auf den S3-Bucket zugegriffen werden kann, muss Ihr Bucket-Name die Zeichenfolge enthalten **greengrass** und der Bucket muss sich in derselben Region befinden, die Sie für verwenden AWS IoT Greengrass. Wählen Sie einen eindeutigen Namen (wie z. B.: **greengrass-bucket-user-id-epoch-time**). Verwenden Sie keinen Punkt (.) im Bucket-Namen.

7. Suchen Sie auf der Registerkarte „AWS IoT Greengrass Konsole“ Ihren S3-Bucket und wählen Sie ihn aus. Suchen Sie Ihre hochgeladene Datei `squeezenet.zip` und wählen Sie Select (Auswählen). Möglicherweise müssen Sie die Refresh (Aktualisieren) wählen, um die Liste der verfügbaren Buckets und Dateien zu aktualisieren.
8. Geben Sie für Destination path (Zielpfad) `/greengrass-machine-learning/mxnet/squeezenet` ein.

Dies ist das Ziel für das lokale Modell im Lambda-Laufzeit-Namespace. Wenn Sie die Gruppe bereitstellen, AWS IoT Greengrass ruft das Quellmodellpaket ab und extrahiert dann den Inhalt in das angegebene Verzeichnis. Die Lambda-Beispielfunktion für dieses Tutorial ist bereits für die Verwendung dieses Pfads (in der `model_path` Variablen) konfiguriert.

9. Wählen Sie unter Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Keine Systemgruppe aus.
10. Wählen Sie Add resource (Ressource hinzufügen) aus.

Verwenden von SageMaker KI-trainierten Modellen

In diesem Tutorial wird ein Modell verwendet, das in Amazon S3 gespeichert ist, aber Sie können auch problemlos SageMaker KI-Modelle verwenden. Die AWS IoT Greengrass Konsole verfügt über eine integrierte SageMaker KI-Integration, sodass Sie diese Modelle nicht manuell auf Amazon S3 hochladen müssen. Die Anforderungen und Einschränkungen für die Verwendung von SageMaker KI-Modellen finden Sie unter [the section called “Unterstützte Modellquellen”](#).

So verwenden Sie ein SageMaker KI-Modell:

- Wählen Sie als Modellquelle die Option Ein in AWS SageMaker KI trainiertes Modell verwenden und wählen Sie dann den Namen des Trainingsjobs des Modells aus.
- Geben Sie unter Zielpfad den Pfad zu dem Verzeichnis ein, in dem Ihre Lambda-Funktion nach dem Modell sucht.

Schritt 7: Hinzufügen eines Abonnements zur Greengrass-Gruppe

In diesem Schritt fügen Sie der Gruppe ein Abonnement hinzu. Dieses Abonnement ermöglicht es der Lambda-Funktion, Vorhersageergebnisse zu senden, AWS IoT indem sie sie in einem MQTT-Thema veröffentlicht.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.
2. Konfigurieren Sie auf der Seite mit den Abonnementdetails die Quelle und das Ziel wie folgt:
 - a. Wählen Sie unter Quelltyp die Option Lambda-Funktion und dann aus `greengrassObjectClassification`.
 - b. Wählen Sie unter Zieltyp die Option Service und dann IoT Cloud aus.

3. Geben Sie **hello/world** im Themenfilter den Text ein und wählen Sie dann Abonnement erstellen aus.

Schritt 8: Bereitstellen der Greengrass-Gruppe

In diesem Schritt stellen Sie die aktuelle Version der Gruppendefinition für das Greengrass Core-Gerät bereit. Die Definition enthält die Lambda-Funktion, die Ressourcen und die Abonnementkonfigurationen, die Sie hinzugefügt haben.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/1.11.6/bin/daemon` enthält, dann wird der Daemon ausgeführt.

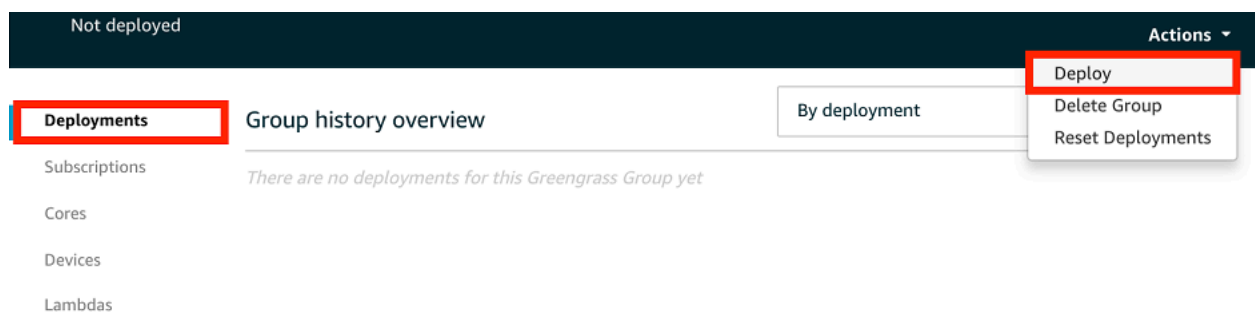
Note

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Deploy aus.



3. Wählen Sie auf der Registerkarte Lambda-Funktionen im Abschnitt System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
4. Wählen Sie im Dialogfeld IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben aus.
5. Wählen Sie Speichern.

Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Eine automatische Erkennung wird empfohlen, unterstützt aber AWS IoT Greengrass auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

 Note

Wenn Sie dazu aufgefordert werden, erteilen Sie die Erlaubnis, die [Greengrass-Servicerolle zu erstellen und sie Ihrer AWS-Konto aktuellen AWS-Region Rolle zuzuordnen](#). Diese Rolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Diensten.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss sollte der für die Bereitstellung angezeigte Status Abgeschlossen lauten.

Weitere Informationen über Bereitstellungen finden Sie unter [Stellen Sie AWS IoT Greengrass Gruppen bereit](#). Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Schritt 9: Testen der Inferenzanwendung

Nun können Sie prüfen, ob die Bereitstellung korrekt konfiguriert ist. Zum Testen abonnieren Sie das hello/world Thema und sehen sich die Prognoseergebnisse an, die von der Lambda-Funktion veröffentlicht werden.

 Note

Wenn ein Bildschirm mit dem Raspberry Pi verbunden ist, wird das Livebild der Kamera in einem Vorschaufenster angezeigt.

1. Wählen Sie in der AWS IoT Konsole unter Test die Option MQTT-Testclient aus.
2. Verwenden Sie für Subscriptions (Abonnements) die folgenden Werte:
 - Verwenden Sie als Abonnementthema „hello/world“.
 - Wählen Sie unter Zusätzliche Konfiguration für die Anzeige von MQTT-Payloads die Option Payloads als Zeichenketten anzeigen aus.
3. Wählen Sie Subscribe (Abonnieren) aus.

Wenn der Test erfolgreich ist, werden die Meldungen der Lambda-Funktion unten auf der Seite angezeigt. Jede Nachricht enthält die ersten fünf der gesamten Prognoseergebnisse des Abbildes im Format: Wahrscheinlichkeit, prognostizierte Klassen-ID und zugehöriger Klassenname.

The screenshot shows the AWS IoT Greengrass console interface for testing MQTT subscriptions. On the left, there's a sidebar with 'Subscribe to a topic' and 'Publish to a topic' buttons. The 'hello/world' topic is selected. The main area shows a 'Publish' section with a text input containing 'hello/world' and a 'Publish to topic' button. Below this is a code editor showing a JSON message:

```
{ "message": "Hello from AWS IoT console" }
```

. At the bottom, a list of received messages is displayed, each with a timestamp and a 'New Prediction' section. The predictions are highlighted with a red box.

Topic	Timestamp	Predictions
hello/world	Mar 30, 2018 1:47:07 PM -0700	New Prediction: [(0.31046376, 'n03637318 lampshade, lamp shade'), (0.11445289, 'n04380533 table lamp'), (0.04436367, 'n04254120 soap dispenser'), (0.035816364, 'n04286575 spotlight, spot'), (0.028093718, 'n03201208 dining table, board')]
hello/world	Mar 30, 2018 1:47:01 PM -0700	New Prediction: [(0.16117829, 'n03876231 paintbrush'), (0.13750333, 'n04442312 toaster'), (0.081819646, 'n03924679 photocopier'), (0.068144165, 'n02783161 ballpoint, ballpoint pen, ballpen, Biro'), (0.044701375, 'n04209239 shower curtain')]
hello/world	Mar 30, 2018 1:46:55 PM -0700	New Prediction: [(0.46284258, 'n04442312 toaster'), (0.16061385, 'n03908618 pencil box, pencil case'), (0.043834824, 'n03291819 envelope'), (0.027529096, 'n03908714 pencil sharpener'), (0.027273422, 'n04209239 shower curtain')]

Problembehandlung bei AWS IoT Greengrass ML-Inferenz

Wenn der Test nicht erfolgreich ist, können Sie folgende Schritte ausführen, um den Fehler zu beheben. Führen Sie im Raspberry Pi-Terminal die Befehle aus.

Fehlerprotokolle prüfen

1. Wechseln Sie zum Root-Benutzer und navigieren Sie zum Verzeichnis `log`. Für den Zugriff auf AWS IoT Greengrass Protokolle sind Root-Rechte erforderlich.

```
sudo su
```



```
cd /greengrass/ggc/var/log
```

- Überprüfen Sie im system-Verzeichnis `runtime.log` oder `python_runtime.log`.

Überprüfen Sie im user/*region/account-id*-Verzeichnis `greengrassObjectClassification.log`.

Weitere Informationen finden Sie unter [the section called "Fehlerbehebung mit Protokollen"](#).

Fehler beim Entpacken in "Runtime.log"

Wenn `runtime.log` einen Fehler enthält, der wie folgt aussieht, stellen Sie sicher, dass Ihr `tar.gz`-Quellmodellpaket über ein übergeordnetes Verzeichnis verfügt.

```
Greengrass deployment error: unable to download the artifact model-arn: Error while processing.  
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /greengrass/ggc/deployment/path/model-arn,  
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/squeezenet_v1.1-0000.params: no such file or directory
```

Wenn Ihr Paket über kein übergeordnetes Verzeichnis verfügt, das die Modelldateien enthält, verwenden Sie den folgenden Befehl, um das Modell erneut zu verpacken:

```
tar -zcvf model.tar.gz ./model
```

Beispiel:

```
#$ tar -zcvf test.tar.gz ./test  
./test  
./test/some.file  
./test/some.file2  
./test/some.file3
```

Note

Verwenden Sie keine abschließenden `/*`-Zeichen in diesem Befehl.

Stellen Sie sicher, dass die Lambda-Funktion erfolgreich bereitgestellt wurde

1. Listet den Inhalt des bereitgestellten Lambda im `/lambda` Verzeichnis auf. Ersetzen Sie die Platzhalterwerte, bevor Sie den Befehl ausführen.

```
cd /greengrass/ggc/deployment/lambda/  
arn:aws:lambda:region:account:function:function-name:function-version  
ls -la
```

2. Überprüfen Sie, ob das Verzeichnis denselben Inhalt wie das `greengrassObjectClassification.zip`-Bereitstellungspaket enthält, das Sie in [Schritt 4: Eine Lambda-Funktion erstellen und veröffentlichen](#) hochgeladen haben.

Stellen Sie sicher, dass sich die `.py`-Dateien und -Abhängigkeiten im Stammverzeichnis befinden.

Sicherstellen, dass das Inferenzmodell erfolgreich bereitgestellt wird

1. Finden Sie die Prozessidentifikationsnummer (PID) des Lambda-Laufzeitprozesses:

```
ps aux | grep 'lambda-function-name*
```

In der Ausgabe erscheint die PID in der zweiten Spalte der Zeile für den Lambda-Laufzeitprozess.

2. Geben Sie den Lambda-Runtime-Namespace ein. Achten Sie darauf, den *pid* Platzhalterwert zu ersetzen, bevor Sie den Befehl ausführen.

Note

Dieses Verzeichnis und sein Inhalt befinden sich im Lambda-Runtime-Namespace, sodass sie in einem regulären Linux-Namespace nicht sichtbar sind.

```
sudo nsenter -t pid -m /bin/bash
```

3. Listen Sie den Inhalt des lokalen Verzeichnisses, das Sie für die ML-Ressource angegeben haben, auf.

```
cd /greengrass-machine-learning/mxnet/squeezenet/  
ls -ls
```

Die Dateien sollten folgendermaßen aussehen:

```
32 -rw-r--r-- 1 ggc_user ggc_group 31675 Nov 18 15:19 synset.txt  
32 -rw-r--r-- 1 ggc_user ggc_group 28707 Nov 18 15:19 squeezenet_v1.1-symbol.json  
4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19  
squeezenet_v1.1-0000.params
```

Nächste Schritte

Erkunden Sie als Nächstes andere Inferenz-Apps. AWS IoT Greengrass stellt weitere Lambda-Funktionen bereit, mit denen Sie lokale Inferenz ausprobieren können. Sie finden das Beispieldatensatzpaket im Ordner der vorkompilierten Bibliotheken, den Sie unter [the section called “Installieren Sie das MXNet Framework”](#) heruntergeladen haben.

Konfigurieren eines Intel Atom

Um dieses Tutorial auf einem Intel Atom-Gerät auszuführen, müssen Sie Quellbilder bereitstellen, die Lambda-Funktion konfigurieren und eine weitere lokale Gerätereource hinzufügen. Um die GPU für die Inferenz zu verwenden, stellen Sie sicher, dass die folgende Software auf Ihrem Gerät installiert ist:

- OpenCL Version 1.0 oder höher
- Python 3.7 und pip

Note

Wenn Ihr Gerät mit Python 3.6 vorkonfiguriert ist, können Sie stattdessen einen Symlink zu Python 3.7 erstellen. Weitere Informationen finden Sie unter [Step 2](#).

- [NumPy](#)
- [OpenCV on Wheels](#)

1. Laden Sie statische PNG- oder JPG-Bilder für die Lambda-Funktion herunter, um sie für die Bildklassifizierung zu verwenden. Das Beispiel funktioniert am besten mit kleinen Bilddateien.

Speichern Sie Ihre Bilddateien in dem Verzeichnis mit der `greengrassObjectClassification.py`-Datei (oder in einem Unterverzeichnis dieses Verzeichnisses). Dies ist im Lambda-Funktionsbereitstellungspaket enthalten, in [the section called “Eine Lambda-Funktion erstellen und veröffentlichen”](#) das Sie hochladen.

Note

Wenn Sie die Kamera verwenden AWS DeepLens, können Sie die integrierte Kamera verwenden oder Ihre eigene Kamera montieren, um Rückschlüsse auf aufgenommene Bilder statt auf statische Bilder zu ziehen. Wir empfehlen jedoch nachdrücklich, zunächst mit statischen Bildern zu beginnen.

Wenn Sie eine Kamera verwenden, stellen Sie sicher, dass das `awscam`-APT-Paket installiert und auf dem neuesten Stand ist. Weitere Informationen finden Sie im AWS DeepLens Entwicklerhandbuch unter [Ihr AWS DeepLens Gerät aktualisieren](#).

2. Wenn Sie Python 3.7 nicht verwenden, stellen Sie sicher, dass Sie einen Symlink von Python 3.x zu Python 3.7 erstellen. Dadurch wird Ihr Gerät für die Verwendung von Python 3 konfiguriert. AWS IoT Greengrass Führen Sie den folgenden Befehl aus, um Ihre Python-Installation zu suchen:

```
which python3
```

Führen Sie den folgenden Befehl aus, um den Symlink zu erstellen.

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

Starten Sie das Gerät neu.

3. Bearbeiten Sie die Konfiguration der Lambda-Funktion. Folgen Sie dem Verfahren unter [the section called “Fügen Sie die Lambda-Funktion zur Gruppe hinzu”](#).

Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die

GPU und Kamera Ihres Geräts zugreifen, ohne die Gerätere Ressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Systembenutzer und Gruppe die Option. **Another user ID/group ID**. Geben Sie als Systembenutzer-ID ein `0`. Geben Sie als Systemgruppen-ID ein `0`.

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

 Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Kein Container aus.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter. [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Aktualisieren Sie den Wert für Timeout auf 5 Sekunden. Dies stellt sicher, dass die Zeit für die Anforderung nicht zu schnell überschritten wird. Nach der Einrichtung dauert es einige Minuten, bis die Inferenz ausgeführt wird.
- Wählen Sie unter Fixiert die Option True aus.
- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis `/sys` die Option Aktiviert aus.
- Wählen Sie für Lambda-Lebenszyklus die Option Diese Funktion langlebig einstellen und unbegrenzt ausführen aus.

b. Um stattdessen im containerisierten Modus zu starten:

 Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Aktualisieren Sie den Wert für Timeout auf 5 Sekunden. Dies stellt sicher, dass die Zeit für die Anforderung nicht zu schnell überschritten wird. Nach der Einrichtung dauert es einige Minuten, bis die Inferenz ausgeführt wird.
 - Wählen Sie für Pinned die Option True aus.
 - Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.
4. Wenn es im containerisierten Modus ausgeführt wird, fügen Sie die erforderliche lokale Geräteressource hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren.

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die GPU Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen.

- a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen aus.
- b. Wählen Sie Lokale Ressource hinzufügen aus.
- c. Definieren Sie die Ressource:
 - Geben Sie für Resource name (Ressourcenname) **renderD128** ein.
 - Wählen Sie als Ressourcentyp die Option Lokales Gerät aus.
 - Geben Sie im Feld Device path (Gerätepfad) **/dev/dri/renderD128** ein.
 - Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.
 - Gewähren Sie für Zugehörigkeiten zu Lambda-Funktionen Lese- und Schreibzugriff auf Ihre Lambda-Funktion.

Konfiguration eines NVIDIA Jetson TX2

Um dieses Tutorial auf einem NVIDIA Jetson auszuführen TX2, stellen Sie Quellimages bereit und konfigurieren Sie die Lambda-Funktion. Bei Verwendung der GPU müssen Sie zusätzlich lokale Gerätere Ressourcen hinzufügen.

1. Stellen Sie sicher, dass Ihr Jetson-Gerät so konfiguriert ist, dass Sie die AWS IoT Greengrass Core-Software installieren können. Weitere Informationen zur Konfiguration des Geräts finden Sie unter [the section called “Einrichten anderer Geräte”](#).
2. Öffnen Sie die MXNet Dokumentation, gehen Sie zu [Installation MXNet auf einem Jetson](#) und folgen Sie den Anweisungen zur Installation MXNet auf dem Jetson-Gerät.

Note

Wenn Sie MXNet aus dem Quellcode bauen möchten, folgen Sie den Anweisungen zum Erstellen der gemeinsam genutzten Bibliothek. Bearbeiten Sie die folgenden Einstellungen in Ihrer `config.mk` Datei, um mit einem TX2 Jetson-Gerät zu arbeiten:

- Fügen Sie `-gencode arch=compute-62, code=sm_62` der `CUDA_ARCH`-Einstellung hinzu.
- Schalten Sie CUDA ein.

```
USE_CUDA = 1
```

3. Laden Sie statische PNG- oder JPG-Bilder für die Lambda-Funktion herunter, um sie für die Bildklassifizierung zu verwenden. Die App funktioniert am besten mit kleinen Bilddateien. Alternativ können Sie eine Kamera auf der Jetson-Karte instrumentieren, um die Quellbilder zu erfassen.

Speichern Sie Ihre Abbilddateien in dem Verzeichnis, das die `greengrassObjectClassification.py`-Datei enthält. Sie können sie auch in einem Unterverzeichnis dieses Verzeichnisses speichern. Dieses Verzeichnis befindet sich im Lambda-Funktionsbereitstellungspaket, in [the section called “Eine Lambda-Funktion erstellen und veröffentlichen”](#) das Sie hochladen.

4. Erstellen Sie einen Symlink von Python 3.7 nach Python 3.6, um Python 3 zu AWS IoT Greengrass verwenden. Führen Sie den folgenden Befehl aus, um Ihre Python-Installation zu suchen:

```
which python3
```

Führen Sie den folgenden Befehl aus, um den Symlink zu erstellen.

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

Starten Sie das Gerät neu.

5. Stellen Sie sicher, dass das `ggc_user` Systemkonto das MXNet Framework verwenden kann:

```
"sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

6. Bearbeiten Sie die Konfiguration der Lambda-Funktion. Folgen Sie dem Verfahren unter [the section called "Fügen Sie die Lambda-Funktion zur Gruppe hinzu"](#).

Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne die Gerätere Ressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

- a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Systembenutzer und Gruppe die Option. **Another user ID/group ID**. Geben Sie als Systembenutzer-ID ein `0`. Geben Sie als Systemgruppen-ID ein `0`.

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called "Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe"](#).

Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Kein Container aus.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis `/sys` die Option Aktiviert aus.
- Fügen Sie unter Umgebungsvariablen die folgenden Schlüssel-Wert-Paare zu Ihrer Lambda-Funktion hinzu. Dadurch wird die Verwendung des Frameworks konfiguriert AWS IoT Greengrass . MXNet

Key (Schlüssel)	Value (Wert)
PATH	/usr/local/cuda/bin: \$PATH
MXNET_HOME	\$HOME/mxnet/
PYTHONPATH	\$MXNET_HOME/python:\$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	\$LD_LIBRARY_PATH:\${CUDA_HOME}/lib64

- b. Um stattdessen im containerisierten Modus zu laufen:

Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Erhöhen Sie den Wert für Speicherlimit. Verwenden Sie 500 MB für die CPU oder mindestens 2000 MB für die GPU.
- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.
- Fügen Sie unter Umgebungsvariablen die folgenden Schlüssel-Wert-Paare zu Ihrer Lambda-Funktion hinzu. Dadurch wird die Verwendung des Frameworks konfiguriert AWS IoT Greengrass . MXNet

Key (Schlüssel)	Value (Wert)
PATH	/usr/local/cuda/bin: \$PATH
MXNET_HOME	\$HOME/mxnet/
PYTHONPATH	\$MXNET_HOME/python:\$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	\$LD_LIBRARY_PATH:\${CUDA_HOME}/lib64

7. Wenn es im containerisierten Modus läuft, fügen Sie die folgenden lokalen Geräteressourcen hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren. Folgen Sie dem Verfahren unter [the section called “Hinzufügen von Ressourcen zur Gruppe”](#).

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die GPU Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen.

Für jede Ressource:

- Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
- Wählen Sie für Systemgruppenbesitzer- und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe automatisch hinzufügen aus, der die Ressource gehört.

Name	Gerätepfad
nvhost-strg	/dev/nvhost-strg
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/ nvhost-ctrl-gpu
nvhost-dbg-gpu	/entwicklung/ nvhost-dbg-gpu
nvhost-prof-gpu	/entwicklung/ nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. Wenn es im containerisierten Modus läuft, fügen Sie die folgende lokale Volume-Ressource hinzu, um Zugriff auf Ihre Gerätekamera zu gewähren. Folgen Sie dem Verfahren unter [the section called “Hinzufügen von Ressourcen zur Gruppe”](#).

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die Kamera Ihres Geräts zugreifen, ohne die Volume-Ressourcen zu konfigurieren.

- Wählen Sie für Resource type (Ressourcentyp) die Option Volume aus.
- Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe automatisch hinzufügen aus, der die Ressource gehört.

Name	Quellpfad	Zielpfad
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

So konfigurieren Sie eine optimierte Inferenz für maschinelles Lernen mit dem AWS-Managementkonsole

Um die Schritte in diesem Tutorial ausführen zu können, müssen Sie AWS IoT Greengrass Core v1.10 oder höher verwenden.

Sie können den SageMaker AI Neo Deep-Learning-Compiler verwenden, um die Prognoseeffizienz nativer Inferenzmodelle für maschinelles Lernen in Tensorflow, Apache MXNet PyTorch, ONNX und XGBoost Frameworks zu optimieren und so einen geringeren Platzbedarf und eine schnellere Leistung zu erzielen. Anschließend können Sie das optimierte Modell herunterladen, die SageMaker AI Neo Deep Learning-Runtime installieren und sie für schnellere Inferenzen auf Ihren AWS IoT Greengrass Geräten bereitstellen.

In diesem Tutorial wird beschrieben, wie Sie eine Greengrass-Gruppe für die AWS-Managementkonsole Ausführung eines Lambda-Inferenzbeispiels verwenden, das Bilder von einer Kamera lokal erkennt, ohne Daten an die Cloud zu senden. Das Inferenzbeispiel greift auf das Kameramodul auf einem Raspberry Pi zu. In diesem Tutorial laden Sie ein vorgefertigtes Modell herunter, das von Resnet-50 trainiert und im Neo Deep Learning-Compiler optimiert wurde. Anschließend verwenden Sie das Modell, um eine lokale Bildklassifizierung auf Ihrem Gerät durchzuführen. AWS IoT Greengrass

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Konfigurieren des Raspberry Pi](#)
2. [Installieren Sie die Neo Deep Learning Runtime](#)
3. [Erstellen Sie eine Inferenz-Lambda-Funktion](#)
4. [Fügen Sie die Lambda-Funktion zur Gruppe hinzu](#)
5. [Hinzufügen einer Neo-optimierten Modellressource zur Gruppe](#)
6. [Hinzufügen Ihrer Kamerageräte-Ressource zur Gruppe](#)

7. [Hinzufügen von Abonnements zur Gruppe](#)
8. [Bereitstellen der Gruppe](#)
9. [Testen des Beispiels](#)

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Raspberry Pi 4 Modell B oder Raspberry Pi 3 Modell B/B+, eingerichtet und konfiguriert für die Verwendung mit AWS IoT Greengrass. Um Ihren Raspberry Pi mit AWS IoT Greengrass einzurichten, führen Sie das [Greengrass Device Setup-Skript](#) aus oder stellen Sie sicher, dass Sie [Modul 1](#) und [Modul 2](#) von [Erste Schritte mit AWS IoT Greengrass](#) abgeschlossen haben.

Note

Der Raspberry Pi benötigt möglicherweise eine [2,5-A-Stromversorgung](#), um die Deep-Learning-Frameworks auszuführen, die normalerweise für die Bildklassifizierung verwendet werden. Ein Netzteil mit einer niedrigeren Nennleistung kann dazu führen, dass das Gerät neu gestartet wird.

- [Raspberry Pi Kameramodul V2 – 8 Megapixel, 1080p](#). Weitere Informationen zum Einrichten der Kamera finden Sie unter [Verbinden der Kamera](#) in der Raspberry Pi-Dokumentation.
- Eine Greengrass-Gruppe und ein Greengrass Core. Weitere Informationen zum Erstellen einer Greengrass-Gruppe oder eines Greengrass-Cores finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#).

Note

Dieses Tutorial verwendet einen Raspberry Pi, AWS IoT Greengrass unterstützt aber auch andere Plattformen wie [Intel Atom](#) und [NVIDIA Jetson TX2](#). Wenn Sie das Intel Atom-Beispiel verwenden, müssen Sie möglicherweise Python 3.6 anstelle von Python 3.7 installieren. Informationen zur Konfiguration Ihres Geräts für die Installation der AWS IoT Greengrass Core-Software finden Sie unter [the section called “Einrichten anderer Geräte”](#). Für Plattformen von Drittanbietern, die AWS IoT Greengrass dies nicht unterstützen, müssen Sie Ihre Lambda-Funktion im Modus ohne Container ausführen. Um im Modus ohne Container ausgeführt zu werden, müssen Sie Ihre Lambda-Funktion als Root ausführen.

Weitere Informationen erhalten Sie unter [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#) und [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

Schritt 1: Konfigurieren des Raspberry Pi

In diesem Schritt installieren Sie Updates für das Raspbian-Betriebssystem, installieren die Kameramodulsoftware und Python-Abhängigkeiten und aktivieren die Kameraschnittstelle.

Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus.

1. Installieren von Updates für Raspbian.

```
sudo apt-get update  
sudo apt-get dist-upgrade
```

2. Installieren Sie die `picamera`-Schnittstelle für das Kameramodul und andere Python-Bibliotheken, die für dieses Tutorial erforderlich sind.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Überprüfen Sie die Installation:

- Stellen Sie sicher, dass Ihre Python 3.7-Installation `pip` enthält.

```
python3 -m pip
```

Wenn `pip` nicht installiert ist, laden Sie es von der [pip-Website](#) herunter und führen Sie dann den folgenden Befehl aus.

```
python3 get-pip.py
```

- Stellen Sie sicher, dass Ihre Python-Version 3.7 oder höher ist.

```
python3 --version
```

Wenn in der Ausgabe eine frühere Version aufgelistet ist, führen Sie den folgenden Befehl aus.

```
sudo apt-get install -y python3.7-dev
```

- Stellen Sie sicher, dass Setuptools und Picamera erfolgreich installiert wurden.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'  
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Wenn die Ausgabe keine Fehler enthält, ist die Überprüfung erfolgreich.

Note

Wenn die auf Ihrem Gerät installierte ausführbare Python-Datei `python3.7` ist, verwenden Sie `python3.7` anstelle von `python3` für die Befehle in diesem Tutorial. Stellen Sie sicher, dass Ihre pip-Installation der richtigen `python3.7`- oder `python3`-Version zugeordnet ist, um Abhängigkeitsfehler zu vermeiden.

3. Starten Sie den Raspberry Pi neu.

```
sudo reboot
```

4. Öffnen Sie das Raspberry Pi-Konfigurations-Tool.

```
sudo raspi-config
```

5. Verwenden Sie die Pfeiltasten zum Öffnen von Interfacing Options (Verbindungsoptionen) und aktivieren Sie die Kameraschnittstelle. Wenn Sie dazu aufgefordert werden, lassen Sie den Neustart des Geräts zu.
6. Geben Sie den folgenden Befehl ein, um die Einrichtung der Kamera zu testen.

```
raspistill -v -o test.jpg
```

So werden ein Vorschaufenster im Raspberry Pi geöffnet, ein Bild mit dem Namen `test.jpg` in Ihrem aktuellen Verzeichnis gespeichert und Informationen über die Kamera im Raspberry Pi-Terminal angezeigt.

Schritt 2: Installieren Sie die Amazon SageMaker Neo Deep Learning-Laufzeit

Installieren Sie in diesem Schritt die Neo Deep Learning Runtime (DLR) auf Ihrem Raspberry Pi.

Note

Wir empfehlen, Version 1.1.0 für dieses Tutorial zu installieren.

1. Melden Sie sich per Fernzugriff bei Ihrem Raspberry Pi an.

```
ssh pi@your-device-ip-address
```

2. Öffnen Sie die DLR-Dokumentation, öffnen Sie [Installing DLR](#) und suchen Sie die Wheel-URL für Raspberry Pi-Geräte. Folgen Sie dann den Anweisungen, um die DLR auf Ihrem Gerät zu installieren. Sie können beispielsweise pip verwenden:

```
pip3 install rasp3b-wheel-url
```

3. Überprüfen Sie nach der Installation der DLR die folgende Konfiguration:

- Stellen Sie sicher, dass das `ggc_user`-Systemkonto die DLR-Bibliothek verwenden kann.

```
sudo -u ggc_user bash -c 'python3 -c "import dlr"'
```

- Stellen Sie sicher, dass sie installiert NumPy ist.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Schritt 3: Erstellen Sie eine Inferenz-Lambda-Funktion

In diesem Schritt erstellen Sie ein Lambda-Funktions-Bereitstellungspaket und eine Lambda-Funktion. Veröffentlichen Sie anschließend eine Funktionsversion und erstellen einen Alias.

1. Laden Sie auf Ihrem Computer das DLR-Beispiel für Raspberry Pi von [the section called "Beispiele für Machine Learning"](#) herunter.
2. Entpacken Sie die heruntergeladene `dlr-py3-armv7l.tar.gz`-Datei.


```
cd path-to-downloaded-sample  
tar -xvzf dlr-py3-armv7l.tar.gz
```

Das `examples`-Verzeichnis im extrahierten Beispelpaket enthält Funktionscode und Abhängigkeiten.

- `inference.py` ist der in diesem Tutorial verwendete Inferenzcode. Sie können diesen Code als Vorlage verwenden, um eine eigene Inferenzfunktion zu erstellen.
- `greengrasssdk` ist Version 1.5.0 des AWS IoT Greengrass Core SDK für Python.

 Note

Wenn eine neue Version verfügbar ist, können Sie sie herunterladen und die SDK-Version in Ihrem Bereitstellungspaket aktualisieren. Weitere Informationen finden Sie unter [AWS IoT Greengrass Core SDK for Python](#) on GitHub.

3. Komprimieren Sie den Inhalt des `examples`-Verzeichnisses in eine Datei namens `optimizedImageClassification.zip`. Dies ist Ihr Bereitstellungspaket.

```
cd path-to-downloaded-sample/dlr-py3-armv7l/examples  
zip -r optimizedImageClassification.zip .
```

Das Bereitstellungspaket enthält den Funktionscode und die Abhängigkeiten. Dazu gehört der Code, der die Neo-Deep-Learning-Laufzeit Python aufruft, APIs um Inferenzen mit den Neo-Deep-Learning-Compilermodellen durchzuführen.

 Note

Stellen Sie sicher, dass sich die `.py`-Dateien und -Abhängigkeiten im Stammverzeichnis befinden.

4. Fügen Sie nun die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzu.

Wählen Sie auf der Lambda-Konsolenseite Funktionen und dann Funktion erstellen aus.

5. Wählen Sie Author von Grund auf neu und verwenden Sie die folgenden Werte, um Ihre Funktion zu erstellen:

- Geben Sie für Function name (Funktionsname) **optimizedImageClassification** ein.
- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.

The screenshot shows the 'Basic information' tab in the AWS Lambda console. The 'Function name' field contains 'optimizedImageClassification'. The 'Runtime' dropdown menu is set to 'Python 3.7'. The 'Permissions' section shows a link to 'Choose or create an execution role'. At the bottom right, there are 'Cancel' and 'Create function' buttons, with the latter highlighted by a red rectangle.

6. Wählen Sie Funktion erstellen.

Laden Sie jetzt Ihr Lambda-Funktionsbereitstellungspaket hoch und registrieren Sie den Handler.

1. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.

The screenshot shows the 'Code source' tab in the AWS Lambda console. The 'Upload from' dropdown menu is open, and the option '.zip file' is selected and circled in red. Other options include 'Amazon S3 location'. The 'Test' button is also visible.

2. Wählen Sie Ihr **optimizedImageClassification.zip** Bereitstellungspaket und dann Speichern aus.

3. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.

- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
- Geben Sie unter Handler **inference.handler** ein.

Wählen Sie Speichern.

Veröffentlichen Sie als Nächstes die erste Version Ihrer Lambda-Funktion. Anschließend erstellen Sie einen [Alias für die Version](#).

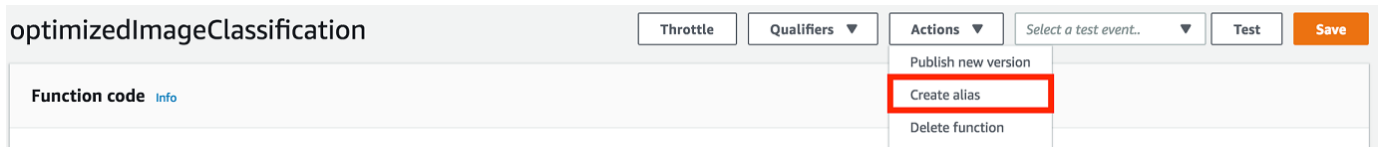
Note

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

1. Wählen Sie im Menü Actions die Option Publish new version aus.

2. Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.

3. Wählen Sie auf der Konfigurationsseite `optimizedImageClassification: 1` im Menü Aktionen die Option Alias erstellen aus.



4. Geben Sie auf der Seite Create a new alias folgende Werte an:
 - Geben Sie unter Name **m1Test0pt** ein.
 - Geben Sie in Version (Version) **1** ein.

Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

5. Wählen Sie Erstellen aus.

Fügen Sie nun die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzu.

Schritt 4: Fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu

Fügen Sie in diesem Schritt die Lambda-Funktion zur Gruppe hinzu und konfigurieren Sie dann ihren Lebenszyklus.

Fügen Sie zunächst die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzu.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen und dann Hinzufügen aus.
3. Wählen Sie die Lambda-Funktion und wählen Sie `optimizedImageClassification`.
4. Wählen Sie in der Lambda-Funktionsversion den Alias für die Version aus, die Sie veröffentlicht haben.

Als Nächstes konfigurieren Sie den Lebenszyklus der Lambda-Funktion.

1. Nehmen Sie im Abschnitt Lambda-Funktionskonfiguration die folgenden Aktualisierungen vor.

Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Systembenutzer und Gruppe die Option. **Another user ID/group ID**. Geben Sie als Systembenutzer-ID ein **0**. Geben Sie als Systemgruppen-ID ein **0**.

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Kein Container aus.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Geben Sie in Timeout (Zeitüberschreitung) **10 seconds** ein.
- Wählen Sie für Pinned die Option True aus.

Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis `/sys` die Option Aktiviert aus.

b. Um stattdessen im containerisierten Modus zu starten:

Fügen Sie die Lambda-Funktion zur Gruppe hinzu

 Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Wählen Sie für Systembenutzer und Gruppe die Option Gruppenstandard verwenden aus.
- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Gruppenstandard verwenden aus.
- Geben Sie in Memory Limit (Speicherlimit) **1024 MB** ein.
- Geben Sie in Timeout (Zeitüberschreitung) **10 seconds** ein.
- Wählen Sie für Pinned die Option True aus.

Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.

2. Wählen Sie Lambda-Funktion hinzufügen.

Schritt 5: Fügen Sie der SageMaker Greengrass-Gruppe eine KI-NEO-optimierte Modellressource hinzu

In diesem Schritt erstellen Sie eine Ressource für das optimierte ML-Inferenzmodell und laden Sie sie in einen Amazon S3 S3-Bucket hoch. Suchen Sie dann das hochgeladene Amazon S3 S3-Modell in der AWS IoT Greengrass Konsole und verknüpfen Sie die neu erstellte Ressource mit der Lambda-Funktion. Dies ermöglicht es der Funktion, auf ihre Ressourcen auf dem Core-Gerät zuzugreifen.

1. Navigieren Sie auf Ihrem Computer zum `resnet50`-Verzeichnis im Beispieldatensatz, das Sie in [the section called “Erstellen Sie eine Inferenz-Lambda-Funktion”](#) entpackt haben.

 Note

Wenn Sie das NVIDIA Jetson-Beispiel verwenden, müssen Sie stattdessen das `resnet18`-Verzeichnis im Beispieldatensatz verwenden. Weitere Informationen finden Sie unter [the section called “Konfiguration eines NVIDIA Jetson TX2”](#).

```
cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
```

Dieses Verzeichnis enthält vorkompilierte Modellartefakte für ein mit Resnet-50 trainiertes Bildklassifikationsmodell.

2. Komprimieren Sie die Dateien innerhalb des `resnet50`-Verzeichnisses in eine Datei namens `resnet50.zip`.

```
zip -r resnet50.zip .
```

3. Wählen Sie auf der Gruppenkonfigurationsseite für Ihre AWS IoT Greengrass Gruppe den Tab Ressourcen aus. Navigieren Sie zum Abschnitt Machine Learning und wählen Sie Add machine learning resource (Maschinelle Lernressource hinzufügen) aus. Geben Sie auf der Seite Eine maschinelle Lernressource erstellen für Ressourcenname **resnet50_model** ein.
4. Wählen Sie als Modellquelle die Option Ein in S3 gespeichertes Modell verwenden aus, z. B. ein mit dem Deep Learning Compiler optimiertes Modell.
5. Wählen Sie unter S3-URI die Option Browse S3 aus.

Note

Derzeit werden optimierte SageMaker KI-Modelle automatisch in Amazon S3 gespeichert. Mit dieser Option können Sie Ihr optimiertes Modell in Ihrem Amazon S3 S3-Bucket finden. Weitere Informationen zur Modelloptimierung in SageMaker KI finden Sie in der [SageMaker AI Neo-Dokumentation](#).

6. Wählen Sie Upload a model (Ein Modell hochladen).
7. Laden Sie auf der Registerkarte Amazon S3 S3-Konsole Ihre ZIP-Datei in einen Amazon S3 S3-Bucket hoch. Weitere Informationen finden Sie unter [Wie lade ich Dateien und Ordner in einen S3-Bucket hoch?](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Note

Ihr Bucket-Name muss die Zeichenfolge **greengrass** enthalten. Wählen Sie einen eindeutigen Namen (wie z. B.: **greengrass-dlr-bucket-*user-id*-epoch-time**). Verwenden Sie keinen Punkt (.) im Bucket-Namen.

8. Suchen Sie auf der Registerkarte „AWS IoT Greengrass Konsole“ Ihren Amazon S3 S3-Bucket und wählen Sie ihn aus. Suchen Sie Ihre hochgeladene Datei `resnet50.zip` und wählen Sie Select (Auswählen). Möglicherweise müssen Sie die Seite aktualisieren, um die Liste der verfügbaren Buckets und Dateien zu aktualisieren.
9. Geben Sie im Feld Zielpfad den Wert ein `/ml_model`.

Local path

`/ml_model`

Dies ist das Ziel für das lokale Modell im Lambda-Laufzeit-Namespace. Wenn Sie die Gruppe bereitstellen, AWS IoT Greengrass ruft das Quellmodellpaket ab und extrahiert dann den Inhalt in das angegebene Verzeichnis.

Note

Wir empfehlen Ihnen dringend, den genauen Pfad für Ihren lokalen Pfad zu verwenden. Die Verwendung eines anderen lokalen Modellzielverzeichnisses in diesem Schritt führt dazu, dass einige in diesem Tutorial enthaltene Befehle zur Fehlerbehebung ungenau sind. Wenn Sie einen anderen Pfad verwenden, müssen Sie eine `MODEL_PATH`-Umgebungsvariable einrichten, die genau den von Ihnen hier angegebenen Pfad verwendet. Weitere Informationen zu Umgebungsvariablen finden Sie unter [AWS Lambda -Umgebungsvariablen](#).

10. Bei Ausführung im containerisierten Modus:
 - a. Wählen Sie unter Systemgruppenbesitzer- und Dateizugriffsberechtigungen die Option Systemgruppe und Berechtigungen angeben aus.
 - b. Wählen Sie Nur-Lese-Zugriff und anschließend Ressource hinzufügen aus.

Schritt 6: Hinzufügen einer Kamerageräte-Ressource zur Greengrass-Gruppe

Erstellen Sie in diesem Schritt eine Ressource für das Kameramodul und verknüpfen Sie sie mit der Lambda-Funktion. Dadurch kann die Lambda-Funktion auf die Ressource auf dem Kerngerät zugreifen.

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne diese Gerätereource zu konfigurieren.

1. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Registerkarte Ressourcen aus.
2. Wählen Sie auf der Registerkarte Lokale Ressourcen die Option Lokale Ressource hinzufügen aus.
3. Verwenden Sie auf der Seite Lokale Ressource hinzufügen die folgenden Werte:
 - Geben Sie für Resource name (Ressourcenname) **videoCoreSharedMemory** ein.
 - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
 - Geben Sie für Lokaler Gerätepfad den Wert ein **/dev/vcsm**.

Der Gerätepfad ist der lokale absolute Pfad der Gerätereource. Dieser Pfad kann nur auf ein zeichenorientiertes Gerät oder Blockgerät unter `/dev` verweisen.

- Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.

Mit der Option Group owner file access permission (Dateizugriffsberechtigung des Gruppenbesitzers) können Sie zusätzliche Dateizugriffsberechtigungen für den Lambda-Prozess erteilen. Weitere Informationen finden Sie unter [Dateizugriffsberechtigung des Gruppenbesitzers](#).

4. Wählen Sie unten auf der Seite die Option Ressource hinzufügen aus.
5. Erstellen Sie auf der Registerkarte Ressourcen eine weitere lokale Ressource, indem Sie Hinzufügen wählen und die folgenden Werte verwenden:
 - Geben Sie für Resource name (Ressourcenname) **videoCoreInterface** ein.
 - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
 - Geben Sie für Lokaler Gerätepfad den Wert ein **/dev/vchiq**.
 - Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.
6. Wählen Sie Add resource (Ressource hinzufügen) aus.

Schritt 7: Hinzufügen von Abonnements zur Greengrass-Gruppe

In diesem Schritt fügen Sie der Gruppe Abonnements hinzu. Diese Abonnements ermöglichen es der Lambda-Funktion, Vorhersageergebnisse zu senden, AWS IoT indem sie sie in einem MQTT-Thema veröffentlichen.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.
2. Konfigurieren Sie auf der Seite Abonnement erstellen die Quelle und das Ziel wie folgt:
 - a. Wählen Sie unter Quelltyp die Option Lambda-Funktion und dann aus `optimizedImageClassification`.
 - b. Wählen Sie unter Zieltyp die Option Service und dann IoT Cloud aus.
 - c. Geben Sie im Themenfilter Folgendes ein **/resnet-50/predictions** und wählen Sie dann Abonnement erstellen aus.
3. Hinzufügen eines zweiten Abonnements. Wählen Sie die Registerkarte Abonnements aus, wählen Sie Abonnement hinzufügen und konfigurieren Sie Quelle und Ziel wie folgt:
 - a. Wählen Sie unter Quelltyp die Option Dienste und dann IoT Cloud aus.
 - b. Wählen Sie unter Zieltyp die Option Lambda-Funktion und dann aus `optimizedImageClassification`.
 - c. Geben Sie **/resnet-50/test** im Themenfilter Folgendes ein und wählen Sie dann Create subscription aus.

Schritt 8: Bereitstellen der Greengrass-Gruppe

In diesem Schritt stellen Sie die aktuelle Version der Gruppendefinition für das Greengrass Core-Gerät bereit. Die Definition enthält die Lambda-Funktion, die Ressourcen und die Abonnementkonfigurationen, die Sie hinzugefügt haben.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen `root`-Eintrag für `/greengrass/ggc/packages/latest-core-version/bin/daemon` enthält, dann wird der Daemon ausgeführt.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option `Deploy` aus.
3. Wählen Sie auf der Registerkarte `Lambda-Funktionen` die Option `IP-Detektor` und dann `Bearbeiten` aus.
4. Wählen Sie im Dialogfeld `IP-Detektoreinstellungen bearbeiten` die Option `MQTT-Broker-Endpunkte automatisch erkennen` und überschreiben aus und wählen Sie `Speichern` aus.

Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Eine automatische Erkennung wird empfohlen, unterstützt aber AWS IoT Greengrass auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

Note

Wenn Sie dazu aufgefordert werden, erteilen Sie die Erlaubnis, die [Greengrass-Servicerolle zu erstellen und sie Ihrer AWS-Konto aktuellen AWS-Region Rolle zuzuordnen](#). Diese Rolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Diensten.

Auf der Seite `Deployments` werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss sollte der für die Bereitstellung angezeigte Status `Abgeschlossen` lauten.

Weitere Informationen über Bereitstellungen finden Sie unter [Stellen Sie AWS IoT Greengrass Gruppen bereit](#). Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Testen des Inferenzbeispiels

Nun können Sie prüfen, ob die Bereitstellung korrekt konfiguriert ist. Um zu testen, abonnieren Sie das Thema `/resnet-50/predictions` und veröffentlichen Sie jede Nachricht zum `/resnet-50/`

test-Thema. Dadurch wird die Lambda-Funktion ausgelöst, um ein Foto mit Ihrem Raspberry Pi aufzunehmen und Rückschlüsse auf das aufgenommene Bild zu ziehen.

 Note

Wenn Sie das NVIDIA Jetson-Beispiel verwenden, stellen Sie sicher, dass Sie stattdessen die Themen `resnet-18/predictions` und `resnet-18/test` verwenden.

 Note

Wenn ein Bildschirm mit dem Raspberry Pi verbunden ist, wird das Livebild der Kamera in einem Vorschaufenster angezeigt.

1. Wählen Sie auf der Startseite der AWS IoT Konsole unter Test die Option MQTT-Testclient aus.
2. Wählen Sie für Abonnements die Option Thema abonnieren aus. Verwenden Sie die folgenden Werte. Übernehmen Sie für die verbleibenden Optionen die Standardwerte.
 - Geben Sie für Abonnementthema **/resnet-50/predictions** ein.
 - Wählen Sie unter Zusätzliche Konfiguration für die Anzeige von MQTT-Payloads die Option Payloads als Zeichenketten anzeigen aus.
3. Wählen Sie Subscribe (Abonnieren) aus.
4. Wählen Sie In einem Thema veröffentlichen, geben Sie es **/resnet-50/test** als Themennamen ein und wählen Sie Veröffentlichen.
5. Wenn der Test erfolgreich ist, veranlasst die veröffentlichte Nachricht die Raspberry Pi-Kamera, ein Bild aufzunehmen. Eine Meldung der Lambda-Funktion wird unten auf der Seite angezeigt. Diese Meldung enthält das Vorhersageergebnis des Bildes im Format: vorhergesagter Klassenname, Wahrscheinlichkeit und maximaler Speicherverbrauch.

Konfigurieren eines Intel Atom

Um dieses Tutorial auf einem Intel Atom-Gerät auszuführen, müssen Sie Quellbilder bereitstellen, die Lambda-Funktion konfigurieren und eine weitere lokale Gerätereource hinzufügen. Um die GPU für die Inferenz zu verwenden, stellen Sie sicher, dass die folgende Software auf Ihrem Gerät installiert ist:

- OpenCL Version 1.0 oder höher
- Python 3.7 und pip
- [NumPy](#)
- [OpenCV on Wheels](#)

1. Laden Sie statische PNG- oder JPG-Bilder für die Lambda-Funktion herunter, um sie für die Bildklassifizierung zu verwenden. Das Beispiel funktioniert am besten mit kleinen Bilddateien.

Speichern Sie Ihre Bilddateien in dem Verzeichnis mit der `inference.py`-Datei (oder in einem Unterverzeichnis dieses Verzeichnisses). Dies ist im Lambda-Funktionsbereitstellungspaket enthalten, in [the section called “Erstellen Sie eine Inferenz-Lambda-Funktion”](#) das Sie hochladen.

 Note

Wenn Sie die Kamera verwenden AWS DeepLens, können Sie die integrierte Kamera verwenden oder Ihre eigene Kamera montieren, um Rückschlüsse auf aufgenommene Bilder statt auf statische Bilder zu ziehen. Wir empfehlen jedoch nachdrücklich, zunächst mit statischen Bildern zu beginnen.

Wenn Sie eine Kamera verwenden, stellen Sie sicher, dass das `awscam`-APT-Paket installiert und auf dem neuesten Stand ist. Weitere Informationen finden Sie im AWS DeepLens Entwicklerhandbuch unter [Ihr AWS DeepLens Gerät aktualisieren](#).

2. Bearbeiten Sie die Konfiguration der Lambda-Funktion. Folgen Sie dem Verfahren unter [the section called “Fügen Sie die Lambda-Funktion zur Gruppe hinzu”](#).

 Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne die Gerätere Ressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

- a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Systembenutzer und Gruppe die Option. **Another user ID/group ID**
Geben Sie als Systembenutzer-ID ein **0**. Geben Sie als Systemgruppen-ID ein **0**.

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

 Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Kein Container aus.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter. [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Erhöhen Sie den Wert für Timeout auf 2 Minuten. Dies stellt sicher, dass die Zeit für die Anforderung nicht zu schnell überschritten wird. Nach der Einrichtung dauert es einige Minuten, bis die Inferenz ausgeführt wird.
- Wählen Sie für Pinned die Option True aus.
- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis `/sys` die Option Aktiviert aus.

b. Um stattdessen im containerisierten Modus zu starten:

 Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Erhöhen Sie den Wert für Speicherlimit auf 3000 MB.
- Erhöhen Sie den Wert für Timeout auf 2 Minuten. Dies stellt sicher, dass die Zeit für die Anforderung nicht zu schnell überschritten wird. Nach der Einrichtung dauert es einige Minuten, bis die Inferenz ausgeführt wird.

- Wählen Sie für Pinned die Option True aus.
 - Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.
3. Fügen Sie Ihrer Neo-optimierten Modellressource der Gruppe hinzu. Laden Sie die Modellressourcen in das `resnet50`-Verzeichnis des Beispieldpakets hoch, das Sie in [the section called “Erstellen Sie eine Inferenz-Lambda-Funktion”](#) entpackt haben. Dieses Verzeichnis enthält vorkompilierte Modellartefakte für ein mit Resnet-50 trainiertes Bildklassifikationsmodell. Folgen Sie dem Verfahren unter [the section called “Hinzufügen einer Neo-optimierten Modellressource zur Gruppe”](#) mit den folgenden Updates:
- Komprimieren Sie die Dateien innerhalb des `resnet50`-Verzeichnisses in eine Datei namens `resnet50.zip`.
 - Geben Sie auf der Seite Eine maschinelle Lernressource erstellen für Ressourcenname **`resnet50_model`** ein.
 - Hochladen der `resnet50.zip`-Datei
4. Wenn es im containerisierten Modus ausgeführt wird, fügen Sie die erforderliche lokale Geräteressource hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren.

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die GPU Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen.

- a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen aus.
- b. Wählen Sie im Abschnitt Lokale Ressourcen die Option Lokale Ressource hinzufügen aus.
- c. Definieren Sie die Ressource:
 - Geben Sie für Resource name (Ressourcenname) **`renderD128`** ein.
 - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.
 - Geben Sie für Lokaler Gerätepfad den Wert ein **`/dev/dri/renderD128`**.
 - Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe, der die Ressource gehört, automatisch hinzufügen aus.

Konfiguration eines NVIDIA Jetson TX2

Um dieses Tutorial auf einem NVIDIA Jetson auszuführen TX2, stellen Sie Quellimages bereit, konfigurieren Sie die Lambda-Funktion und fügen Sie weitere lokale Geräteressourcen hinzu.

1. Stellen Sie sicher, dass Ihr Jetson-Gerät so konfiguriert ist, dass Sie die AWS IoT Greengrass Core-Software installieren und die GPU als Inferenz verwenden können. Weitere Informationen zur Konfiguration des Geräts finden Sie unter [the section called “Einrichten anderer Geräte”](#). Um die GPU für Inferenz auf einem NVIDIA Jetson zu verwenden TX2, müssen Sie CUDA 10.0 und cuDNN 7.0 auf Ihrem Gerät installieren, wenn Sie Ihr Board mit Jetpack 4.3 abbilden.
2. Laden Sie statische PNG- oder JPG-Bilder für die Lambda-Funktion herunter, um sie für die Bildklassifizierung zu verwenden. Das Beispiel funktioniert am besten mit kleinen Bilddateien.

Speichern Sie Ihre Abbilddateien in dem Verzeichnis, das die `inference.py`-Datei enthält. Sie können sie auch in einem Unterverzeichnis dieses Verzeichnisses speichern. Dieses Verzeichnis befindet sich im Lambda-Funktionsbereitstellungspaket, in [the section called “Erstellen Sie eine Inferenz-Lambda-Funktion”](#) das Sie hochladen.

Note

Sie können auf Wunsch stattdessen eine Kamera auf der Jetson-Karte instrumentieren, um die Quellbilder zu erfassen. Wir empfehlen jedoch nachdrücklich, zunächst mit statischen Bildern zu beginnen.

3. Bearbeiten Sie die Konfiguration der Lambda-Funktion. Folgen Sie dem Verfahren unter [the section called “Fügen Sie die Lambda-Funktion zur Gruppe hinzu”](#).

Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

- a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Ausführen als. **Another user ID/group ID** Geben Sie **0** für UID ein. Geben Sie für GUID ein. **0**

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

 Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Kein Container aus.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter. [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Erhöhen Sie den Wert für Timeout auf 5 Minuten. Dies stellt sicher, dass die Zeit für die Anforderung nicht zu schnell überschritten wird. Nach der Einrichtung dauert es einige Minuten, bis die Inferenz ausgeführt wird.
- Wählen Sie für Pinned die Option True aus.
- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis `/sys` die Option Aktiviert aus.

b. Um stattdessen im containerisierten Modus zu starten:

 Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Erhöhen Sie den Wert für Speicherlimit. Für die Verwendung des bereitgestellten Modells im GPU-Modus verwenden Sie mindestens 2000 MB.

- Erhöhen Sie den Wert für Timeout auf 5 Minuten. Dies stellt sicher, dass die Zeit für die Anforderung nicht zu schnell überschritten wird. Nach der Einrichtung dauert es einige Minuten, bis die Inferenz ausgeführt wird.
 - Wählen Sie für Pinned die Option True aus.
 - Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.
4. Fügen Sie Ihrer Neo-optimierten Modellressource der Gruppe hinzu. Laden Sie die Modellressourcen in das `resnet18`-Verzeichnis des Beispieldpakets hoch, das Sie in [the section called “Erstellen Sie eine Inferenz-Lambda-Funktion”](#) entpackt haben. Dieses Verzeichnis enthält vorkompilierte Modellartefakte für ein mit Resnet-18 geschultes Bildklassifikationsmodell. Folgen Sie dem Verfahren unter [the section called “Hinzufügen einer Neo-optimierten Modellressource zur Gruppe”](#) mit den folgenden Updates:
- Komprimieren Sie die Dateien innerhalb des `resnet18`-Verzeichnisses in eine Datei namens `resnet18.zip`.
 - Geben Sie auf der Seite Eine maschinelle Lernressource erstellen für Ressourcenname **`resnet18_model`** ein.
 - Hochladen der `resnet18.zip`-Datei
5. Wenn das Gerät im Containermodus ausgeführt wird, fügen Sie die erforderlichen lokalen Geräteressourcen hinzu, um Zugriff auf die GPU Ihres Geräts zu gewähren.

 Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die GPU Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen.

- a. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen aus.
- b. Wählen Sie im Abschnitt Lokale Ressourcen die Option Lokale Ressource hinzufügen aus.
- c. Definieren Sie die einzelnen Ressourcen:
 - Verwenden Sie für Resource name (Ressourcenname) und Device path (Gerätepfad) die Werte in der folgenden Tabelle. Erstellen Sie für jede Zeile in der Tabelle genau eine Geräteressource.
 - Wählen Sie in Resource type (Ressourcentyp) die Option Device (Gerät) aus.

- Wählen Sie für Systemgruppenbesitzer- und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe automatisch hinzufügen aus, der die Ressource gehört.

Name	Gerätepfad
nvhost-strg	/dev/nvhost-strg
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/ nvhost-ctrl-gpu
nvhost-dbg-gpu	/entwicklung/ nvhost-dbg-gpu
nvhost-prof-gpu	/entwicklung/ nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. Wenn es im containerisierten Modus läuft, fügen Sie die folgende lokale Volume-Ressource hinzu, um Zugriff auf Ihre Gerätekamera zu gewähren. Folgen Sie dem Verfahren unter [the section called “Hinzufügen einer Neo-optimierten Modellressource zur Gruppe”](#).

Note

Wenn Sie im Modus ohne Container arbeiten, AWS IoT Greengrass können Sie auf die Kamera Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen.

- Wählen Sie für Resource type (Ressourcentyp) die Option Volume aus.
- Wählen Sie für Systemgruppenbesitzer und Dateizugriffsberechtigungen die Option Dateisystemberechtigungen der Systemgruppe automatisch hinzufügen aus, der die Ressource gehört.

Name	Quellpfad	Zielpfad
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

- Aktualisieren Sie Ihre Gruppenabonnements, um das richtige Verzeichnis zu verwenden. Folgen Sie dem Verfahren unter [the section called “Hinzufügen von Abonnements zur Gruppe”](#) mit den folgenden Updates:
 - Geben Sie für den ersten Themenfilter **/resnet-18/predictions** ein.
 - Geben Sie für den zweiten Themenfilter **/resnet-18/test** ein.
- Aktualisieren Sie Ihre Testabonnements, um das richtige Verzeichnis zu verwenden. Folgen Sie dem Verfahren unter [the section called “Testen des Beispiels”](#) mit den folgenden Updates:
 - Wählen Sie für Abonnements die Option Thema abonnieren aus. Geben Sie für Abonnementthema **/resnet-18/predictions** ein.
 - Geben Sie auf der Seite **/resnet-18/predictions** das **/resnet-18/test**-Thema an, zu dem veröffentlicht werden soll.

Problembehandlung bei AWS IoT Greengrass ML-Inferenz

Wenn der Test nicht erfolgreich ist, können Sie folgende Schritte ausführen, um den Fehler zu beheben. Führen Sie im Raspberry Pi-Terminal die Befehle aus.

Fehlerprotokolle prüfen

- Wechseln Sie zum Root-Benutzer und navigieren Sie zum Verzeichnis `log`. Für den Zugriff auf AWS IoT Greengrass Protokolle sind Root-Rechte erforderlich.

```
sudo su
cd /greengrass/ggc/var/log
```

- Prüfen Sie `runtime.log`, ob Fehler vorliegen.

```
cat system/runtime.log | grep 'ERROR'
```

Sie können auch in Ihrem benutzerdefinierten Lambda-Funktionsprotokoll nach Fehlern suchen:

```
cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
```

Weitere Informationen finden Sie unter [the section called “Fehlerbehebung mit Protokollen”](#).

Stellen Sie sicher, dass die Lambda-Funktion erfolgreich bereitgestellt wurde

1. Listet den Inhalt des bereitgestellten Lambda im `/lambda` Verzeichnis auf. Ersetzen Sie die Platzhalterwerte, bevor Sie den Befehl ausführen.

```
cd /greengrass/ggc/deployment/lambda/  
arn:aws:lambda:region:account:function:function-name:function-version  
ls -la
```

2. Überprüfen Sie, ob das Verzeichnis denselben Inhalt wie das `optimizedImageClassification.zip`-Bereitstellungspaket enthält, das Sie in [Schritt 3: Erstellen Sie eine Inferenz-Lambda-Funktion](#) hochgeladen haben.

Stellen Sie sicher, dass sich die `.py`-Dateien und -Abhängigkeiten im Stammverzeichnis befinden.

Stellen Sie sicher, dass das Inferenzmodell erfolgreich bereitgestellt wird

1. Finden Sie die Prozessidentifikationsnummer (PID) des Lambda-Laufzeitprozesses:

```
ps aux | grep lambda-function-name
```

In der Ausgabe erscheint die PID in der zweiten Spalte der Zeile für den Lambda-Laufzeitprozess.

2. Geben Sie den Lambda-Runtime-Namespace ein. Achten Sie darauf, den `pid` Platzhalterwert zu ersetzen, bevor Sie den Befehl ausführen.

 Note

Dieses Verzeichnis und sein Inhalt befinden sich im Lambda-Runtime-Namespace, sodass sie in einem regulären Linux-Namespace nicht sichtbar sind.

```
sudo nsenter -t pid -m /bin/bash
```

3. Listen Sie den Inhalt des lokalen Verzeichnisses, das Sie für die ML-Ressource angegeben haben, auf.

 Note

Wenn Ihr ML-Ressourcenpfad anders als `m1_model` lautet, müssen Sie ihn hier ersetzen.

```
cd /m1_model  
ls -ls
```

Die Dateien sollten folgendermaßen aussehen:

```
56 -rw-r--r-- 1 ggc_user ggc_group    56703 Oct 29 20:07 model.json  
196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params  
256 -rw-r--r-- 1 ggc_user ggc_group    261848 Oct 29 20:07 model.so  
32 -rw-r--r-- 1 ggc_user ggc_group     30564 Oct 29 20:08 synset.txt
```

Die Lambda-Funktion kann **`/dev/dri/renderD128`** nicht finden.

Dies kann passieren, wenn OpenCL keine Verbindung zu den benötigten GPU-Geräten herstellen kann. Sie müssen Gerätere Ressourcen für die erforderlichen Geräte für Ihre Lambda-Funktion erstellen.

Nächste Schritte

Betrachten Sie als Nächstes weitere optimierte Modelle. Informationen finden Sie in der [SageMaker AI Neo-Dokumentation](#).

Datenströme auf dem AWS IoT Greengrass Core verwalten

AWS IoT Greengrass Stream Manager macht es einfacher und zuverlässiger, großvolumige IoT-Daten auf den AWS Cloud zu übertragen. Stream Manager verarbeitet Datenströme lokal und exportiert sie AWS Cloud automatisch in die. Diese Funktion lässt sich in gängige Szenarien wie maschinelles Lernen (ML) -Inferenz integrieren, bei der Daten lokal verarbeitet und analysiert werden, bevor sie an die AWS Cloud oder an lokale Speicherziele exportiert werden.

Der Stream-Manager vereinfacht die Anwendungsentwicklung. Ihre IoT-Anwendungen können einen standardisierten Mechanismus verwenden, um Datenstreams mit hohem Volumen zu verarbeiten und lokale Datenaufbewahrungsrichtlinien zu verwalten, anstatt benutzerdefinierte Funktionen für die Verwaltung von Streams zu erstellen. IoT-Anwendungen können Streams lesen und darin schreiben. Sie können Richtlinien für Speichertyp, Größe und Datenaufbewahrung pro Stream definieren, um zu steuern, wie der Stream-Manager Streams verarbeitet und exportiert.

Der Stream-Manager ist für den Einsatz in Umgebungen mit intermittierender oder eingeschränkter Konnektivität konzipiert. Sie können die Bandbreitenverwendung, das Timeoutverhalten und die Verarbeitung von Streamdaten definieren, wenn der Core verbunden oder getrennt ist. Für kritische Daten können Sie Prioritäten festlegen, um die Reihenfolge zu steuern, in der Streams in die AWS Cloud exportiert werden.

Sie können automatische Exporte in die zur Speicherung oder AWS Cloud zur weiteren Verarbeitung und Analyse konfigurieren. Stream Manager unterstützt den Export in die folgenden AWS Cloud Ziele.

- Kanäle rein AWS IoT Analytics. AWS IoT Analytics ermöglicht es Ihnen, erweiterte Analysen Ihrer Daten durchzuführen, um Geschäftsentscheidungen zu treffen und Modelle für maschinelles Lernen zu verbessern. Weitere Informationen finden Sie unter [Was ist AWS IoT Analytics?](#) im AWS IoT Analytics Benutzerhandbuch.
- Streams in Kinesis Data Streams. Kinesis Data Streams wird häufig verwendet, um große Datenmengen zu aggregieren und sie in ein Data Warehouse oder einen Map-Reduce-Cluster zu laden. Weitere Informationen finden Sie unter [Was ist Amazon Kinesis Data Streams?](#) im Amazon Kinesis Developer Guide.
- Eigenschaften von Vermögenswerten in AWS IoT SiteWise. AWS IoT SiteWise ermöglicht das Sammeln, Organisieren und Analysieren von Daten aus Industrieanlagen in großem Maßstab. Weitere Informationen finden Sie unter [Was ist AWS IoT SiteWise?](#) im AWS IoT SiteWise Benutzerhandbuch.

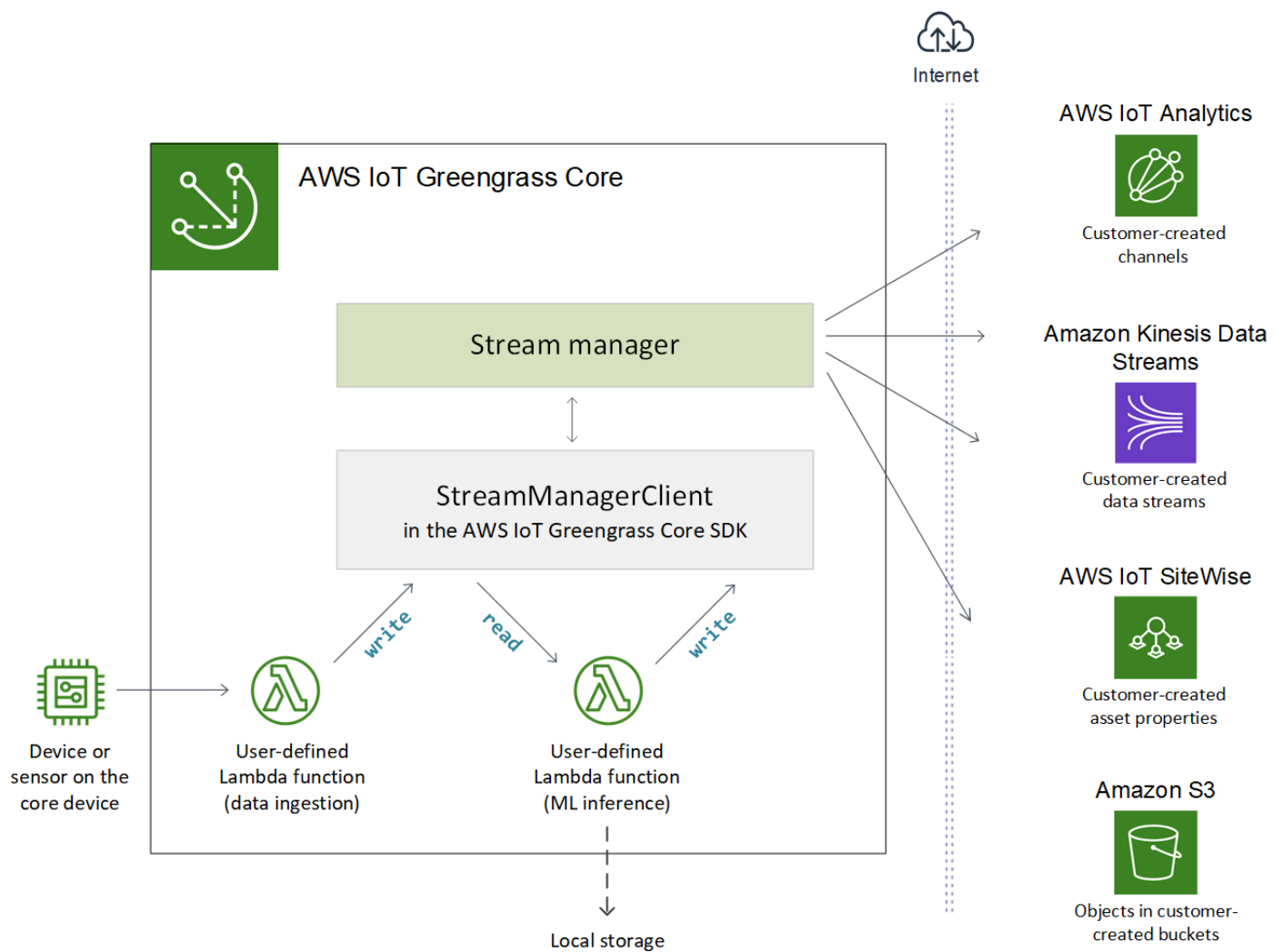
- Objekte in Amazon S3. Sie können Amazon S3 verwenden, um große Datenmengen zu speichern und abzurufen. Weitere Informationen finden Sie unter [Was ist Amazon S3?](#) im Amazon Simple Storage Service Developer Guide.

Stream-Management-Workflow

Ihre IoT-Anwendungen interagieren über das AWS IoT Greengrass Core SDK mit dem Stream Manager. In einem einfachen Workflow verarbeitet eine benutzerdefinierte Lambda-Funktion, die auf dem Greengrass-Kern ausgeführt wird, IoT-Daten wie Temperatur- und Druckmesswerte in Zeitreihen. Die Lambda-Funktion könnte die Daten filtern oder komprimieren und dann das AWS IoT Greengrass Core SDK aufrufen, um die Daten in einen Stream im Stream Manager zu schreiben. Der Stream-Manager kann den Stream auf der Grundlage der für den Stream definierten Richtlinien AWS Cloud automatisch in den exportieren. Benutzerdefinierte Lambda-Funktionen können Daten auch direkt an lokale Datenbanken oder Speicher-Repositorys senden.

Ihre IoT-Anwendungen können mehrere benutzerdefinierte Lambda-Funktionen enthalten, die Streams lesen oder in sie schreiben. Diese lokalen Lambda-Funktionen können Streams lesen und in sie schreiben, um Daten lokal zu filtern, zu aggregieren und zu analysieren. Dies ermöglicht es, schnell auf lokale Ereignisse zu reagieren und wertvolle Informationen zu extrahieren, bevor die Daten vom Core in die Cloud oder lokale Ziele übertragen werden.

Ein Beispielworkflow ist im folgenden Diagramm dargestellt.



Um Stream Manager zu verwenden, konfigurieren Sie zunächst die Stream-Manager-Parameter, um Laufzeiteinstellungen auf Gruppenebene zu definieren, die für alle Streams auf dem Greengrass-Kern gelten. Mit diesen anpassbaren Einstellungen können Sie steuern, wie Stream Manager Streams speichert, verarbeitet und exportiert, basierend auf Ihren Geschäftsanforderungen und Umgebungsbeschränkungen. Weitere Informationen finden Sie unter [the section called “Konfigurieren des Stream-Managers”](#).

Nachdem Sie den Stream Manager konfiguriert haben, können Sie Ihre IoT-Anwendungen erstellen und bereitstellen. Dies sind in der Regel benutzerdefinierte Lambda-Funktionen, die `StreamManagerClient` im AWS IoT Greengrass Core SDK verwendet werden, um Streams zu erstellen und mit ihnen zu interagieren. Während der Stream-Erstellung definiert die Lambda-Funktion Richtlinien pro Stream, wie Exportziele, Priorität und Persistenz. Weitere Informationen, einschließlich Codeausschnitten für Operationen, finden Sie unter [StreamManagerClient the section called “Wird verwendet StreamManagerClient, um mit Streams zu arbeiten”](#).

Tutorials zur Konfiguration eines einfachen Workflows finden Sie unter [the section called “Exportieren von Daten-Streams \(Konsole\)”](#) oder [the section called “Exportieren von Daten-Streams \(CLI\)”](#)

Voraussetzungen

Die folgenden Anforderungen gelten für die Verwendung von Stream Manager:

- Sie müssen die AWS IoT Greengrass Core-Software v1.10 oder höher mit aktiviertem Stream Manager verwenden. Weitere Informationen finden Sie unter [the section called “Konfigurieren des Stream-Managers”](#).

Stream Manager wird auf OpenWrt Distributionen nicht unterstützt.

- Die Java 8-Laufzeitumgebung (JDK 8) muss auf dem Core installiert sein.
 - Führen Sie für Debian-basierte Distributionen (einschließlich Raspbian) oder Ubuntu-basierte Distributionen den folgenden Befehl aus:

```
sudo apt install openjdk-8-jdk
```

- Führen Sie für Red Hat-basierte Distributionen (einschließlich Amazon Linux) den folgenden Befehl aus:

```
sudo yum install java-1.8.0-openjdk
```

Weitere Informationen finden Sie unter [How to download and install prebuilt OpenJDK packages](#) in der OpenJDK-Dokumentation.

- Stream Manager benötigt zusätzlich zu Ihrer Basissoftware AWS IoT Greengrass Core mindestens 70 MB RAM. Ihr gesamter Speicherbedarf hängt von Ihrer Arbeitslast ab.
- Benutzerdefinierte Lambda-Funktionen müssen das [AWS IoT Greengrass Core SDK](#) verwenden, um mit dem Stream Manager zu interagieren. Das AWS IoT Greengrass Core SDK ist in mehreren Sprachen verfügbar, aber nur die folgenden Versionen unterstützen Stream Manager-Operationen:
 - Java SDK (v1.4.0 oder höher)
 - Python SDK (v1.5.0 oder höher)
 - Node.js SDK (v1.6.0 oder höher)

Laden Sie die Version des SDK herunter, die Ihrer Lambda-Funktionslaufzeit entspricht, und nehmen Sie sie in Ihr Lambda-Funktionsbereitstellungspaket auf.

 Note

Das AWS IoT Greengrass Core SDK für Python benötigt Python 3.7 oder höher und hat andere Paketabhängigkeiten. Weitere Informationen finden Sie unter [Erstellen eines Lambda-Funktionsbereitstellungspakets \(Konsole\)](#) oder [Erstellen eines Lambda-Funktionsbereitstellungspakets \(CLI\)](#).

- Wenn Sie AWS Cloud Exportziele für einen Stream definieren, müssen Sie Ihre Exportziele erstellen und Zugriffsberechtigungen in der Greengrass-Gruppenrolle gewähren. Je nach Ziel können auch andere Anforderungen gelten. Weitere Informationen finden Sie unter:
 - [the section called “AWS IoT Analytics Kanäle”](#)
 - [the section called “Amazon Kinesis Kinesis-Datenströme”](#)
 - [the section called “AWS IoT SiteWise Eigenschaften von Vermögenswerten”](#)
 - [the section called “Amazon-S3-Objekte”](#)

Sie sind für die Wartung dieser AWS Cloud Ressourcen verantwortlich.

Datensicherheit

Beachten Sie bei der Verwendung des Stream-Managers die folgenden Sicherheitsüberlegungen.

Lokale Datensicherheit

AWS IoT Greengrass verschlüsselt keine Stream-Daten im Ruhezustand oder bei der lokalen Übertragung zwischen Komponenten auf dem Kerngerät.

- Daten im Ruhezustand. Stream-Daten werden lokal in einem Speicherverzeichnis auf dem Greengrass-Kern gespeichert. Verlässt sich aus Gründen der AWS IoT Greengrass Datensicherheit auf Unix-Dateiberechtigungen und vollständige Festplattenverschlüsselung, sofern aktiviert. Sie können den optionalen Parameter [STREAM_MANAGER_STORE_ROOT_DIR](#) verwenden, um das Speicherverzeichnis anzugeben. Wenn Sie diesen Parameter später ändern, um ein anderes Speicherverzeichnis zu verwenden, werden das vorherige Speicherverzeichnis und sein Inhalt AWS IoT Greengrass nicht gelöscht.

- Daten werden lokal übertragen. AWS IoT Greengrass verschlüsselt keine Stream-Daten bei der lokalen Übertragung auf dem Core zwischen Datenquellen, Lambda-Funktionen, dem AWS IoT Greengrass Core SDK und dem Stream Manager.
- Daten bei der Übertragung zum AWS Cloud Datenströme, die vom Stream Manager exportiert werden, AWS Cloud verwenden die standardmäßige AWS Service-Client-Verschlüsselung mit Transport Layer Security (TLS).

Weitere Informationen finden Sie unter [the section called “Datenverschlüsselung”](#).

Client-Authentifizierung

Stream Manager-Clients verwenden das AWS IoT Greengrass Core SDK, um mit dem Stream Manager zu kommunizieren. Wenn die Client-Authentifizierung aktiviert ist, können nur Lambda-Funktionen in der Greengrass-Gruppe mit Streams im Stream Manager interagieren. Wenn die Clientauthentifizierung deaktiviert ist, kann jeder auf dem Greengrass Core ausgeführte Prozess (z. B. [Docker-Container](#)) mit Streams im Stream-Manager interagieren. Sie sollten die Authentifizierung nur deaktivieren, wenn Ihr Geschäftsfall dies erfordert.

Sie verwenden den Parameter [STREAM_MANAGER_AUTHENTICATE_CLIENT](#), um den Clientauthentifizierungsmodus festzulegen. Sie können diesen Parameter über die Konsole oder AWS IoT Greengrass API konfigurieren. Änderungen werden wirksam, nachdem die Gruppe bereitgestellt wurde.

	Enabled	Disabled
Parameterwert	true (Standard und empfohlen)	false
Zulässige Clients	Benutzerdefinierte Lambda-Funktionen in der Greengrass-Gruppe	Benutzerdefinierte Lambda-Funktionen in der Greengrass-Gruppe

	Enabled	Disabled
		Andere Prozesse, die auf dem Greengrass Core-Gerät ausgeführt werden

Weitere Informationen finden Sie auch unter

- [the section called “Konfigurieren des Stream-Managers”](#)
- [the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”](#)
- [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#)
- [the section called “Exportieren von Daten-Streams \(Konsole\)”](#)
- [the section called “Exportieren von Daten-Streams \(CLI\)”](#)

AWS IoT Greengrass Streammanager konfigurieren

Im AWS IoT Greengrass Kern kann Stream Manager IoT-Gerätedaten speichern, verarbeiten und exportieren. Der Stream-Manager stellt Parameter bereit, die Sie zum Konfigurieren von Laufzeiteinstellungen auf Gruppenebene verwenden. Diese Einstellungen gelten für alle Streams auf dem Greengrass-Core. Sie können die AWS IoT Konsole oder AWS IoT Greengrass API verwenden, um die Stream Manager-Einstellungen zu konfigurieren. Änderungen werden wirksam, nachdem die Gruppe bereitgestellt wurde.

Note

Nachdem Sie den Stream Manager konfiguriert haben, können Sie IoT-Anwendungen erstellen und bereitstellen, die auf dem Greengrass-Kern ausgeführt werden und mit dem Stream Manager interagieren. Bei diesen IoT-Anwendungen handelt es sich in der Regel um benutzerdefinierte Lambda-Funktionen. Weitere Informationen finden Sie unter [the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”](#).

Stream-Manager-Parameter

Der Stream-Manager stellt die folgenden Parameter bereit, mit denen Sie Einstellungen auf Gruppenebene definieren können. Alle Parameter sind optional.

Speicherverzeichnis

Parametername: `STREAM_MANAGER_STORE_ROOT_DIR`

Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern von Streams verwendet wird. Dieser Wert muss mit einem Schrägstrich (z. B. `/data`) beginnen.

Hinweise zum Sichern von Streamdaten finden Sie unter [the section called “Lokale Datensicherheit”](#).

Minimale AWS IoT Greengrass Kernversion: 1.10.0

Server port

Parametername: `STREAM_MANAGER_SERVER_PORT`

Die lokale Portnummer, die für die Kommunikation mit dem Stream-Manager verwendet wird. Der Standardwert ist 8088.

Minimale AWS IoT Greengrass Kernversion: 1.10.0

Client authentifizieren

Parametername: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Gibt an, ob Clients authentifiziert werden müssen, um mit dem Stream-Manager zu interagieren. Die gesamte Interaktion zwischen Clients und Stream Manager wird vom AWS IoT Greengrass Core SDK gesteuert. Dieser Parameter bestimmt, welche Clients das AWS IoT Greengrass Core SDK aufrufen können, um mit Streams zu arbeiten. Weitere Informationen finden Sie unter [the section called “Client-Authentifizierung”](#).

Gültige Werte sind `true` oder `false`. Der Standardwert ist `true` (empfohlen).

- `true`. Erlaubt nur Greengrass Lambda-Funktionen als Clients. Lambda-Funktionsclients verwenden interne AWS IoT Greengrass Kernprotokolle, um sich beim AWS IoT Greengrass Core SDK zu authentifizieren.
- `false`. Ermöglicht es jedem Prozess, der auf dem AWS IoT Greengrass Core läuft, ein Client zu sein. Setzen Sie diese Einstellung nicht auf `false`, es sei denn, Ihr Geschäftsfall erfordert dies. Legen Sie diesen Wert beispielsweise `false` nur dann fest, wenn Nicht-Lambda-

Prozesse auf dem Core-Gerät direkt mit dem Stream-Manager kommunizieren müssen, z. B. [Docker-Container](#), die auf dem Core ausgeführt werden.

Minimale AWS IoT Greengrass Core-Version: 1.10.0

Maximale Bandbreite

Parametername: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

Die durchschnittliche maximale Bandbreite (in Kilobit pro Sekunde), die zum Exportieren von Daten verwendet werden kann. Die Standardeinstellung erlaubt die unbegrenzte Nutzung der verfügbaren Bandbreite.

Minimale AWS IoT Greengrass Kernversion: 1.10.0

Größe des Threadpools

Parametername: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

Die maximale Anzahl der aktiven Threads, die zum Exportieren von Daten verwendet werden können. Der Standardwert ist 5.

Die optimale Größe hängt von der Hardware, dem Stream-Volume und der geplanten Anzahl der Exportstreams ab. Wenn die Exportgeschwindigkeit langsam ist, können Sie diese Einstellung anpassen, um die optimale Größe für Ihre Hardware und Ihren Geschäftsfall zu ermitteln. Die CPU und der Arbeitsspeicher Ihrer Core-Geräte-Hardware sind begrenzende Faktoren. Um zu starten, können Sie versuchen, diesen Wert gleich der Anzahl der Prozessorkerne auf dem Gerät festzulegen.

Achten Sie darauf, keine Größe festzulegen, die höher ist, als Ihre Hardware unterstützen kann. Jeder Stream verbraucht Hardwareressourcen, daher sollten Sie versuchen, die Anzahl der Exportstreams auf eingeschränkten Geräten zu begrenzen.

Minimale AWS IoT Greengrass Kernversion: 1.10.0

JVM-Argumente

Parametername: `JVM_ARGS`

Benutzerdefinierte Java Virtual Machine-Argumente, die beim Start an den Stream-Manager übergeben werden. Mehrere Argumente sollten durch Leerzeichen getrennt werden.

Verwenden Sie diesen Parameter nur, wenn Sie die von der JVM verwendeten Standardeinstellungen außer Kraft setzen müssen. Beispielsweise müssen Sie möglicherweise

die Standard-Heap-Größe erhöhen, wenn Sie eine große Anzahl von Streams exportieren möchten.

Minimale AWS IoT Greengrass Kernversion: 1.10.0

Schreibgeschützte Eingabedateiverzeichnisse

Parametername: `STREAM_MANAGER_READ_ONLY_DIRS`

Eine durch Kommas getrennte Liste absoluter Pfade zu den Verzeichnissen außerhalb des Root-Dateisystems, in denen Eingabedateien gespeichert sind. Stream Manager liest und lädt die Dateien auf Amazon S3 hoch und mountet die Verzeichnisse schreibgeschützt. Weitere Informationen zum Exportieren nach Amazon S3 finden Sie unter [the section called “Amazon-S3-Objekte”](#).

Verwenden Sie diesen Parameter nur, wenn die folgenden Bedingungen zutreffen:

- Das Eingabedateiverzeichnis für einen Stream, der nach Amazon S3 exportiert wird, befindet sich an einem der folgenden Speicherorte:
 - Eine andere Partition als das Root-Dateisystem.
 - Unter `/tmp` im Root-Dateisystem.
- Die [Standardcontainerisierung](#) der Greengrass-Gruppe ist Greengrass-Container.

Beispielwert: `/mnt/directory-1,/mnt/directory-2,/tmp`

Minimale Kernversion AWS IoT Greengrass : 1.11.0

Mindestgröße für mehrteiligen Upload

Parametername:

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

Die Mindestgröße (in Byte) eines Teils in einem mehrteiligen Upload auf Amazon S3. Stream Manager verwendet diese Einstellung und die Größe der Eingabedatei, um zu bestimmen, wie Daten in einer mehrteiligen PUT-Anfrage gebündelt werden. Der Standard- und Mindestwert ist 5242880 Byte (5 MB).

Note

Stream Manager verwendet die `sizeThresholdForMultipartUploadBytes` Eigenschaft des Streams, um zu bestimmen, ob ein- oder mehrteiliger Upload nach Amazon S3 exportiert werden soll. Benutzerdefinierte Lambda-Funktionen legen diesen

Schwellenwert fest, wenn sie einen Stream erstellen, der nach Amazon S3 exportiert wird. Der Standardschwellenwert ist 5 MB.

Minimale AWS IoT Greengrass Core-Version: 1.11.0

Konfigurieren der Stream-Manager-Einstellungen (Konsole)

Sie können die AWS IoT Konsole für die folgenden Verwaltungsaufgaben verwenden:

- [Überprüfen, ob der Stream-Manager aktiviert ist](#)
- [Aktivieren oder Deaktivieren des Stream-Managers während der Gruppenerstellung](#)
- [Aktivieren oder Deaktivieren des Stream-Managers für eine vorhandene Gruppe](#)
- [Ändern der Einstellungen für den Stream Manager](#)

Änderungen werden wirksam, nachdem die Greengrass-Gruppe bereitgestellt wurde. Ein Tutorial, das zeigt, wie Sie eine Greengrass-Gruppe bereitstellen, die eine Lambda-Funktion enthält, die mit dem Stream-Manager interagiert, finden Sie unter. [the section called “Exportieren von Daten-Streams \(Konsole\)”](#)

Note

Wenn Sie die Konsole verwenden, um den Stream Manager zu aktivieren und die Gruppe bereitzustellen, ist die Speichergröße für den Stream Manager standardmäßig auf 4194304 KB (4 GB) festgelegt. Wir empfehlen, die Speichergröße auf mindestens 128000 KB festzulegen.

So prüfen Sie, ob der Stream-Manager aktiviert ist (Konsole)

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie die Registerkarte Lambda-Funktionen.

4. Wählen Sie unter System Lambda-Funktionen die Option Stream Manager und dann Bearbeiten aus.
5. Überprüfen Sie den Status „Aktiviert“ oder „Deaktiviert“. Alle konfigurierten benutzerdefinierten Stream-Manager-Einstellungen werden ebenfalls angezeigt.

So aktivieren oder deaktivieren Sie den Stream-Manager während der Gruppenerstellung (Konsole)

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie Create Group. Ihre Auswahl auf der nächsten Seite bestimmt, wie Sie den Stream-Manager für die Gruppe konfigurieren.
3. Gehen Sie unter Name your Group weiter und wählen Sie eine Greengrass-Kernseite aus.
4. Wählen Sie Create group (Gruppe erstellen) aus.
5. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen, wählen Sie Stream Manager und dann Bearbeiten aus.
 - Um den Stream-Manager mit Standardeinstellungen zu aktivieren, wählen Sie Mit Standardeinstellungen aktivieren.
 - Um den Stream-Manager mit benutzerdefinierten Einstellungen zu aktivieren, wählen Sie Einstellungen anpassen aus.
 1. Wählen Sie auf der Seite Stream Manager konfigurieren die Option Mit benutzerdefinierten Einstellungen aktivieren aus.
 2. Geben Sie unter Benutzerdefinierte Einstellungen Werte für Stream-Manager-Parameter ein. Weitere Informationen finden Sie unter [the section called “Stream-Manager-Parameter”](#). Lassen Sie die Felder leer, AWS IoT Greengrass damit ihre Standardwerte verwendet werden können.
 - Um den Stream-Manager zu deaktivieren, wählen Sie Disable.
 1. Wählen Sie auf der Seite Stream-Manager konfigurieren die Option Deaktivieren aus.

6. Wählen Sie Speichern.
7. Fahren Sie mit den verbleibenden Seiten fort, um Ihre Gruppe zu erstellen.
8. Laden Sie auf der Seite Client-Geräte Ihre Sicherheitsressourcen herunter, überprüfen Sie die Informationen und klicken Sie dann auf Fertig stellen.

 Note

Wenn der Stream-Manager aktiviert ist, müssen Sie [die Java 8-Laufzeitumgebung](#) auf dem Core-Gerät installieren, bevor Sie die Gruppe bereitstellen.

So aktivieren oder deaktivieren Sie den Stream-Manager für eine vorhandene Gruppe (Konsole)

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie die Registerkarte Lambda-Funktionen.
4. Wählen Sie unter System Lambda-Funktionen die Option Stream Manager und dann Bearbeiten aus.
5. Überprüfen Sie den Status „Aktiviert“ oder „Deaktiviert“. Alle konfigurierten benutzerdefinierten Stream-Manager-Einstellungen werden ebenfalls angezeigt.

So ändern Sie die Einstellungen des Stream-Managers (Konsole)

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie die Registerkarte Lambda-Funktionen.

4. Wählen Sie unter System Lambda-Funktionen die Option Stream Manager und dann Bearbeiten aus.
5. Überprüfen Sie den Status „Aktiviert“ oder „Deaktiviert“. Alle konfigurierten benutzerdefinierten Stream-Manager-Einstellungen werden ebenfalls angezeigt.
6. Wählen Sie Speichern.

Konfigurieren der Stream Manager-Einstellungen (CLI)

Verwenden Sie in der die AWS CLIGGStreamManager Lambda-Funktion des Systems, um den Stream-Manager zu konfigurieren. System-Lambda-Funktionen sind Bestandteile der AWS IoT Greengrass Core-Software. Für Stream Manager und einige andere Lambda-Systemfunktionen können Sie die Greengrass-Funktionalität konfigurieren, indem Sie die entsprechenden Function FunctionDefinitionVersion AND-Objekte in der Greengrass-Gruppe verwalten. Weitere Informationen finden Sie unter [the section called “Übersicht über das Gruppenobjektmodell”](#).

Sie können die API für die folgenden Verwaltungsaufgaben verwenden. Die Beispiele in diesem Abschnitt zeigen, wie Sie die verwenden AWS CLI, aber Sie können die AWS IoT Greengrass API auch direkt aufrufen oder ein AWS SDK verwenden.

- [Überprüfen, ob der Stream-Manager aktiviert ist](#)
- [Aktivieren, deaktivieren oder konfigurieren Sie den Stream Manager](#)

Änderungen werden wirksam, nachdem die Gruppe bereitgestellt wurde. Ein Tutorial, das zeigt, wie Sie eine Greengrass-Gruppe mit einer Lambda-Funktion bereitstellen, die mit dem Stream-Manager interagiert, finden Sie unter. [the section called “Exportieren von Daten-Streams \(CLI\)”](#)

Tip

Um zu sehen, ob der Stream-Manager aktiviert ist und von Ihrem Kerngerät aus ausgeführt wird, können Sie den folgenden Befehl in einem Terminal auf dem Gerät ausführen.

```
ps aux | grep -i 'streammanager'
```

So prüfen Sie, ob der Stream-Manager aktiviert ist (CLI)

Stream Manager ist aktiviert, wenn Ihre bereitgestellte Funktionsdefinitionsversion die GGStreamManager Lambda-Funktion des Systems enthält. Zur Überprüfung gehen Sie folgendermaßen vor:

1. Rufen Sie IDs die Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

Sie können diese Werte auch in der AWS IoT Konsole finden. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die LatestVersionWerte und Id aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.
 - Ersetzen Sie *group-id* durch den Id, den Sie kopiert haben.
 - Ersetzen Sie *latest-group-version-id* durch den LatestVersion, den Sie kopiert haben.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. Rufen Sie aus FunctionDefinitionVersionArn der Ausgabe die Version IDs der Funktionsdefinition und der Funktionsdefinition ab.

- Die Funktionsdefinitions-ID ist die GUID, die dem `functions` Segment im Amazon-Ressourcennamen (ARN) folgt.
- Die ID der Funktionsdefinitionsversion ist die GUID, die dem `versions`-Segment im ARN folgt.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/  
functions/function-definition-id/versions/function-definition-version-id
```

5. Rufen Sie die ID der Funktionsdefinitionsversion ab.

- Ersetzen Sie *function-definition-id* durch die Funktionsdefinitions-ID.
- *function-definition-version-id* Ersetzen Sie durch die Versions-ID der Funktionsdefinition.

```
aws greengrass get-function-definition-version \  
--function-definition-id function-definition-id \  
--function-definition-version-id function-definition-version-id
```

Wenn das `functions`-Array in der Ausgabe die `GGStreamManager`-Funktion enthält, ist der Stream-Manager aktiviert. Alle Umgebungsvariablen, die für die Funktion definiert sind, stellen benutzerdefinierte Einstellungen für den Stream-Manager dar.

Um den Stream Manager (CLI) zu aktivieren, zu deaktivieren oder zu konfigurieren

Verwenden Sie in der die `AWS CLI GGStreamManager` Lambda-Funktion des Systems, um den Stream-Manager zu konfigurieren. Änderungen werden wirksam, nachdem Sie die Gruppe bereitgestellt haben.

- Um den Stream-Manager zu aktivieren, schließen Sie `GGStreamManager` in das `functions`-Array Ihrer Funktionsdefinitionsversion ein. Um benutzerdefinierte Einstellungen zu konfigurieren, definieren Sie Umgebungsvariablen für die entsprechenden [Stream-Manager-Parameter](#).
- Um den Stream-Manager zu deaktivieren, entfernen Sie `GGStreamManager` aus dem `functions`-Array Ihrer Funktionsdefinitionsversion.

Stream-Manager mit Standardeinstellungen

Die folgende Beispielkonfiguration aktiviert den Stream-Manager mit Standardeinstellungen. Sie setzt die beliebige Funktions-ID auf `streamManager`.

```
{
  "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
  "FunctionConfiguration": {
    "MemorySize": 4194304,
    "Pinned": true,
    "Timeout": 3
  },
  "Id": "streamManager"
}
```

Note

Was die `FunctionConfiguration` Eigenschaften angeht, wissen Sie vielleicht Folgendes:

- `MemorySize` ist mit den Standardeinstellungen auf 4194304 KB (4 GB) eingestellt. Sie können diesen Wert jederzeit ändern. Wir empfehlen, dass Sie einen `MemorySize` Wert von mindestens 128000 KB angeben.
- muss `Pinned` auf `true` festgelegt sein.
- `Timeout` ist für die Funktionsdefinitionsversion erforderlich, wird vom `GGStreamManager` jedoch nicht verwendet.

Stream-Manager mit benutzerdefinierten Einstellungen

Die folgende Beispielkonfiguration aktiviert Stream Manager mit benutzerdefinierten Werten für die Parameter Speicherverzeichnis, Serverport und Threadpool-Größe.

```
{
  "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
  "FunctionConfiguration": {
    "Environment": {
      "Variables": {
        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
        "STREAM_MANAGER_SERVER_PORT": "1234",

```



```

        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
    },
    "MemorySize": 4194304,
    "Pinned": true,
    "Timeout": 3
},
    "Id": "streamManager"
}

```

AWS IoT Greengrass verwendet Standardwerte für [Stream-Manager-Parameter](#), die nicht als Umgebungsvariablen angegeben sind.

Stream-Manager mit benutzerdefinierten Einstellungen für Amazon S3 S3-Exporte

Die folgende Beispielkonfiguration aktiviert den Stream-Manager mit benutzerdefinierten Werten für das Upload-Verzeichnis und Mindestparametern für die Größe mehrteiliger Uploads.

```

{
    "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",
                "STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
                "10485760"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}

```

Um den Stream Manager (CLI) zu aktivieren, zu deaktivieren oder zu konfigurieren

1. Rufen Sie IDs die Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

Sie können diese Werte auch in der AWS IoT Konsole finden. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die LatestVersionWerte und Id aus der Zielgruppe in die Ausgabe.
3. Rufen Sie die neueste Gruppenversion ab.
 - Ersetzen Sie *group-id* durch den Id, den Sie kopiert haben.
 - Ersetzen Sie *latest-group-version-id* durch den LatestVersion, den Sie kopiert haben.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. Kopieren Sie die CoreDefinitionVersionArn und alle anderen Versionen ARNs aus der Ausgabe, außer FunctionDefinitionVersionArn. Sie verwenden diese Werte später, wenn Sie eine Gruppenversion erstellen.
5. Kopieren Sie die ID der Funktionsdefinition aus FunctionDefinitionVersionArn. Die ID ist die GUID, die dem functions-Segment im ARN folgt, wie im folgenden Beispiel gezeigt.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

Note

Sie können auch eine Funktionsdefinition erstellen, indem Sie den [create-function-definition](#) Befehl ausführen und dann die ID aus der Ausgabe kopieren.

6. Fügen Sie der Funktionsdefinition eine Funktionsdefinitionsversion hinzu.

- *function-definition-id* Ersetzen Sie sie durch Id die, die Sie für die Funktionsdefinition kopiert haben.
- Fügen Sie in das functions Array alle anderen Funktionen ein, die Sie auf dem Greengrass-Kern verfügbar machen möchten. Sie können den `get-function-definition-version` Befehl verwenden, um die Liste der vorhandenen Funktionen abzurufen.

Aktivieren des Stream-Managers mit Standardeinstellungen

Im folgenden Beispiel wird der Stream-Manager aktiviert, indem die `GGStreamManager` Funktion in das functions Array aufgenommen wird. In diesem Beispiel werden Standardwerte für die [Stream-Manager-Parameter](#) verwendet.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
    {
        "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
        "FunctionConfiguration": {
            "MemorySize": 4194304,
            "Pinned": true,
            "Timeout": 3
        },
        "Id": "streamManager"
    },
    {
```

```

        "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
        "FunctionConfiguration": {
            "Executable": "myLambdaFunction.function_handler",
            "MemorySize": 16000,
            "Pinned": true,
            "Timeout": 5
        },
        "Id": "myLambdaFunction"
    },
    ... more user-defined functions
]
}'

```

Note

Die myLambdaFunction Funktion in den Beispielen stellt eine Ihrer benutzerdefinierten Lambda-Funktionen dar.

Aktivieren des Stream-Managers mit benutzerdefinierten Einstellungen

Im folgenden Beispiel wird der Stream-Manager aktiviert, indem die GGStreamManager-Funktion in das functions-Array aufgenommen wird. Alle Stream-Manager-Einstellungen sind optional, es sei denn, Sie möchten die Standardwerte ändern. Dieses Beispiel zeigt, wie Umgebungsvariablen verwendet werden, um benutzerdefinierte Werte festzulegen.

```

aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
    {
        "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
        "FunctionConfiguration": {
            "Environment": {
                "Variables": {
                    "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                    "STREAM_MANAGER_SERVER_PORT": "1234",
                    "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                }
            },
            "MemorySize": 4194304,

```

```

        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
},
{
    "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
    "FunctionConfiguration": {
        "Executable": "myLambdaFunction.function_handler",
        "MemorySize": 16000,
        "Pinned": true,
        "Timeout": 5
    },
    "Id": "myLambdaFunction"
},
... more user-defined functions
]
}'

```

Note

Was die FunctionConfiguration Eigenschaften angeht, wissen Sie vielleicht Folgendes:

- MemorySize ist mit den Standardeinstellungen auf 4194304 KB (4 GB) eingestellt. Sie können diesen Wert jederzeit ändern. Wir empfehlen, dass Sie einen MemorySize Wert von mindestens 128000 KB angeben.
- muss Pinned auf true festgelegt sein.
- Timeout ist für die Funktionsdefinitionsversion erforderlich, wird vom GGStreamManager jedoch nicht verwendet.

Deaktivieren des Stream-Managers

Im folgenden Beispiel wird die GGStreamManager-Funktion weggelassen, wodurch der Stream-Manager deaktiviert wird.

```

aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[

```

```
{
  "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
  "FunctionConfiguration": {
    "Executable": "myLambdaFunction.function_handler",
    "MemorySize": 16000,
    "Pinned": true,
    "Timeout": 5
  },
  "Id": "myLambdaFunction"
},
... more user-defined functions
]
```

 Note

Wenn Sie keine Lambda-Funktionen bereitstellen möchten, können Sie die Version der Funktionsdefinition vollständig weglassen.

7. Kopieren Sie den Arn der Funktionsdefinitionsversion aus der Ausgabe.
8. Erstellen Sie eine Gruppenversion, die die Lambda-Funktion des Systems enthält.
 - Ersetzen Sie *group-id* durch die Id für die Gruppe.
 - *core-definition-version-arn* Ersetzen Sie es durch CoreDefinitionVersionArn das, was Sie aus der neuesten Gruppenversion kopiert haben.
 - *function-definition-version-arn* Ersetzen Sie es durch Arn das, was Sie für die neue Version der Funktionsdefinition kopiert haben.
 - Ersetzen Sie die ARNs für andere Gruppenkomponenten (z. B. SubscriptionDefinitionVersionArn oder DeviceDefinitionVersionArn), die Sie aus der neuesten Gruppenversion kopiert haben.
 - Entfernen Sie alle nicht verwendeten Parameter. Entfernen Sie zum Beispiel --resource-definition-version-arn, wenn Ihre Gruppenversion keine Ressourcen enthält.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
```

```
--device-definition-version-arn device-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Kopieren Sie die Version aus der Ausgabe. Dies ist die ID der neuen Gruppenversion.

10. Stellen Sie die Gruppe mit der neuen Gruppenversion bereit.

- *group-id* Ersetzen Sie sie durch Id die, die Sie für die Gruppe kopiert haben.
- *group-version-id* Ersetzen Sie es durch Version das, was Sie für die neue Gruppenversion kopiert haben.

```
aws greengrass create-deployment \  
--group-id group-id \  
--group-version-id group-version-id \  
--deployment-type NewDeployment
```

Gehen Sie wie folgt vor, wenn Sie die Stream Manager-Einstellungen später erneut bearbeiten möchten. Stellen Sie sicher, dass Sie eine Version der Funktionsdefinition erstellen, die die GGStreamManager Funktion mit der aktualisierten Konfiguration enthält. Die Gruppenversion muss auf alle Komponentenversionen verweisen ARNs , die Sie im Core bereitstellen möchten. Änderungen werden wirksam, nachdem die Gruppe bereitgestellt wurde.

Weitere Informationen finden Sie auch unter

- [Verwalten von Daten-Streams](#)
- [the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”](#)
- [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#)
- [the section called “Exportieren von Daten-Streams \(Konsole\)”](#)
- [the section called “Exportieren von Daten-Streams \(CLI\)”](#)

Wird verwendet StreamManagerClient , um mit Streams zu arbeiten

Benutzerdefinierte Lambda-Funktionen, die auf dem AWS IoT Greengrass Core ausgeführt werden, können das `StreamManagerClient` Objekt im [AWS IoT Greengrass Core SDK](#) verwenden, um Streams im [Stream Manager](#) zu erstellen und dann mit den Streams zu interagieren. Wenn eine Lambda-Funktion einen Stream erstellt, definiert sie die AWS Cloud Ziele, die Priorisierung und andere Export- und Datenaufbewahrungsrichtlinien für den Stream. Um Daten an den Stream Manager zu senden, hängen Lambda-Funktionen die Daten an den Stream an. Wenn ein Exportziel für den Stream definiert ist, exportiert der Stream Manager den Stream automatisch.

Note

In der Regel handelt es sich bei den Clients von Stream Manager um benutzerdefinierte Lambda-Funktionen. Wenn Ihr Geschäftsszenario dies erfordert, können Sie auch zulassen, dass Nicht-Lambda-Prozesse, die auf dem Greengrass-Kern ausgeführt werden (z. B. ein Docker-Container), mit dem Stream Manager interagieren. Weitere Informationen finden Sie unter [the section called “Client-Authentifizierung”](#).

Die Auszüge in diesem Thema zeigen Ihnen, wie Clients Methoden `aufrufenStreamManagerClient`, um mit Streams zu arbeiten. Implementierungsdetails zu den Methoden und ihren Argumenten finden Sie unter den Links zur SDK-Referenz, die hinter jedem Codeausschnitt aufgeführt sind. Tutorials, die eine vollständige Python-Lambda-Funktion enthalten, finden Sie unter [the section called “Exportieren von Daten-Streams \(Konsole\)”](#) oder [the section called “Exportieren von Daten-Streams \(CLI\)”](#).

Ihre Lambda-Funktion sollte `StreamManagerClient` außerhalb des Funktionshandlers instanziiert werden. Wenn sie in dem Handler instanziiert wird, erstellt die Funktion bei jedem Aufruf eine `client` und eine Verbindung zum Stream-Manager.

Note

Wenn Sie `StreamManagerClient` in dem Handler instanziiieren, müssen Sie die `close()`-Methode explizit aufrufen, wenn die `client` seine Arbeit abschließt. Andernfalls hält der `client` die Verbindung offen, und ein anderer Thread läuft, bis das Skript beendet wird.

`StreamManagerClient` unterstützt die folgenden Operationen:

- [the section called “Erstellen eines Nachrichten-Streams”](#)
- [the section called “Anhängen einer Nachricht”](#)
- [the section called “Lesen von Nachrichten”](#)
- [the section called “Auflisten von Streams”](#)
- [the section called “Beschreiben eines Nachrichten-Streams”](#)
- [the section called “Nachrichtenstream aktualisieren”](#)
- [the section called “Löschen eines Nachrichten-Streams”](#)

Erstellen eines Nachrichten-Streams

Um einen Stream zu erstellen, ruft eine benutzerdefinierte Lambda-Funktion die Methode `create` auf und übergibt ein `MessageStreamDefinition` Objekt. Dieses Objekt gibt den eindeutigen Namen für den Stream an und definiert, wie der Stream-Manager mit neuen Daten umgehen soll, wenn die maximale Streamgröße erreicht ist. Sie können mit `MessageStreamDefinition` seinen Datentypen (z. B. `ExportDefinition`, `StrategyOnFull`, und `Persistence`) andere Stream-Eigenschaften definieren. Dazu zählen:

- Das Ziel AWS IoT Analytics, Kinesis Data Streams und Amazon S3 S3-Ziele für automatische Exporte. AWS IoT SiteWise Weitere Informationen finden Sie unter [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#).
- Export-Priorität. Stream-Manager exportiert Streams mit höherer Priorität vor Streams mit niedrigerer Priorität.
- Maximale Batchgröße und Batch-Intervall für AWS IoT Analytics Kinesis Data Streams und AWS IoT SiteWise Ziele. Der Stream-Manager exportiert Nachrichten, wenn eine der Bedingungen erfüllt ist.
- Time-to-live (TTL). Die Zeitspanne, um sicherzustellen, dass die Streamdaten für die Verarbeitung verfügbar sind. Sie sollten sicherstellen, dass die Daten innerhalb dieses Zeitraums verbraucht werden können. Dies ist keine Löschrichtlinie. Die Daten werden möglicherweise nicht unmittelbar nach dem TTL-Zeitraum gelöscht.
- Streampersistenz. Wählen Sie, ob Streams im Dateisystem gespeichert werden sollen, um Daten über Core-Neustarts hinweg zu speichern oder Streams im Speicher zu speichern.
- Startsequenznummer. Geben Sie die Sequenznummer der Nachricht an, die als Startnachricht im Export verwendet werden soll.

Weitere Informationen zu `MessageStreamDefinition` finden Sie in der SDK-Referenz für Ihre Zielsprache:

- [MessageStreamDefinition](#) im Java-SDK
- [MessageStreamDefinition](#) im SDK von Node.js
- [MessageStreamDefinition](#) im Python-SDK

 Note

`StreamManagerClient` bietet auch ein Zielziel, mit dem Sie Streams auf einen HTTP-Server exportieren können. Dieses Ziel dient nur zu Testzwecken. Es ist weder stabil noch wird es für die Verwendung in Produktionsumgebungen unterstützt.

Nachdem ein Stream erstellt wurde, können Ihre Lambda-Funktionen [Nachrichten an den Stream anhängen](#), um Daten für den Export zu senden, und [Nachrichten aus dem Stream für die lokale Verarbeitung lesen](#). Die Anzahl der Streams, die Sie erstellen, hängt von Ihren Hardwarefunktionen und Ihrem Geschäftsfall ab. Eine Strategie besteht darin, für jeden Zielkanal im AWS IoT Analytics oder Kinesis-Datenstream einen Stream zu erstellen. Sie können jedoch mehrere Ziele für einen Stream definieren. Ein Stream hat eine dauerhafte Lebensdauer.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

 Note

Für das Erstellen von Streams mit einem AWS IoT SiteWise oder einem Amazon S3 S3-Exportziel gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.11.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Beispiele

Das folgende Snippet erstellt einen Stream mit dem Namen `StreamName`. Es definiert Stream-Eigenschaften in den `MessageStreamDefinition` und untergeordneten Datentypen.

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 Cloud.
            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-Referenz: [create_message_stream](#) | [MessageStreamDefinition](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().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 Cloud.
            new ExportDefinition()
                .withKinesis(null)
                .withIotAnalytics(null)
                .withIotSitewise(null)
                .withS3TaskExecutor(null)
            )
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

Java-SDK-Referenz: | [createMessageStreamMessageStreamDefinition](#)

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 Cloud.
                    new ExportDefinition()
                        .withKinesis(null)
                        .withIotAnalytics(null)
                        .withIotSitewise(null)
                        .withS3TaskExecutor(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.
});
```

SDK-Referenz für Node.js: [createMessageStream](#) | [MessageStreamDefinition](#)

Weitere Informationen zur Konfiguration von Exportzielen finden Sie unter [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#).

Anhängen einer Nachricht

Um Daten zum Export an den Stream Manager zu senden, hängen Ihre Lambda-Funktionen die Daten an den Zielstream an. Das Exportziel bestimmt den Datentyp, der an diese Methode übergeben werden soll.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Note

Für das Anhängen von Nachrichten mit einem AWS IoT SiteWise oder einem Amazon S3 S3-Exportziel gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Kernversion: 1.11.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Beispiele

AWS IoT Analytics oder Kinesis Data Streams Streams-Exportziele

Das folgende Snippet fügt eine Nachricht an den Stream namens `StreamName` an. Für AWS IoT Analytics unsere Kinesis Data Streams Streams-Ziele hängen Ihre Lambda-Funktionen einen Datenblob an.

Für dieses Snippet gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

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-Referenz: [append_message](#)

Java

```
try (final StreamManagerClient client =
GreengrassClientBuilder.streamManagerClient().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK-Referenz: [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-Referenz: [appendMessage](#)

AWS IoT SiteWise Ziele exportieren

Das folgende Snippet fügt eine Nachricht an den Stream namens StreamName an. Für AWS IoT SiteWise Ziele hängen Ihre Lambda-Funktionen ein PutAssetPropertyValueEntry serialisiertes Objekt an. Weitere Informationen finden Sie unter [the section called “Exportieren nach AWS IoT SiteWise”](#).

Note

Wenn Sie Daten an senden AWS IoT SiteWise, müssen Ihre Daten die Anforderungen der Aktion erfüllen. BatchPutAssetPropertyValue Weitere Informationen finden Sie unter [BatchPutAssetPropertyValue](#) in der AWS IoT SiteWise -API-Referenz.

Für dieses Snippet gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.11.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Python

```
client = StreamManagerClient()
```

```

try:
    # SiteWise requires unique timestamps in all messages. 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",
        data=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-Referenz: [append_message](#) | [PutAssetPropertyValueEntry](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
    in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>();

    // IoTSiteWise requires unique timestamps in all messages. 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 SDK-Referenz: [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-Referenz: [appendMessage | PutAssetPropertyValueEntry](#)

Amazon S3 S3-Exportziele

Das folgende Snippet hängt eine Exportaufgabe an den genannten Stream an. StreamName Für Amazon S3 S3-Ziele hängen Ihre Lambda-Funktionen ein serialisiertes S3ExportTaskDefinition Objekt an, das Informationen über die Quelleingabedatei und das Amazon S3 S3-Zielobjekt enthält. Wenn das angegebene Objekt nicht existiert, erstellt Stream Manager es für Sie. Weitere Informationen finden Sie unter [the section called “Exportieren nach Amazon S3”](#).

Für dieses Snippet gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.11.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

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",
        data=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-Referenz: `append\_message` | `S3 ExportTaskDefinition`](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
        ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}

```

[Java-SDK-Referenz: `appendMessage` | `S3 ExportTaskDefinition`](#)

Node.js

```

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        // Append an Amazon S3 export task definition and print the sequence number.
        const taskDefinition = new S3ExportTaskDefinition()
            .withBucket("BucketName")
            .withKey("KeyName")
            .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
            util.validateAndSerializeToJsonBytes(taskDefinition));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.

```

```
});
```

[SDK-Referenz für Node.js: appendMessage | S3 ExportTaskDefinition](#)

Lesen von Nachrichten

Lesen Sie Nachrichten aus einem Stream.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Beispiele

Das folgende Snippet liest Nachrichten aus dem Stream namens StreamName. Die Read-Methode verwendet ein optionales ReadMessagesOptions-Objekt, das die Sequenznummer angibt, von der aus mit dem Lesen begonnen werden soll, die minimale und maximale Anzahl zu lesen und ein Timeout für das Lesen von Nachrichten.

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-Referenz: [read_messages](#) | [ReadMessagesOptions](#)

Java

```

try (final StreamManagerClient client =
GreengrassClientBuilder.streamManagerClient().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 SDK-Referenz: [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-Referenz: [readMessages](#) | [ReadMessagesOptions](#)

Auflisten von Streams

Ruft die Liste der Streams im Stream-Manager ab.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Beispiele

Das folgende Snippet ruft eine Liste der Streams (nach Namen) im Stream-Manager ab.

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-Referenz: [list_streams](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: [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-Referenz: [ListStreams](#)

Beschreiben eines Nachrichten-Streams

Ruft Metadaten zu einem Stream ab, einschließlich der Stream-Definition, -Größe und des Exportstatus.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Beispiele

Das folgende Snippet ruft Metadaten über den Stream mit dem Namen StreamName ab, einschließlich Definition, Größe und Exporterstatus des Streams.

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-Referenz: [describe_message_stream](#)

Java

```

try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().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-SDK-Referenz: [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.
});
```

SDK-Referenz für Node.js: [describeMessageStream](#)

Nachrichtenstream aktualisieren

Aktualisieren Sie die Eigenschaften eines vorhandenen Streams. Möglicherweise möchten Sie einen Stream aktualisieren, wenn sich Ihre Anforderungen nach der Erstellung des Streams ändern. Beispiel:

- Fügen Sie eine neue [Exportkonfiguration](#) für ein AWS Cloud Ziel hinzu.
- Erhöhen Sie die maximale Größe eines Streams, um zu ändern, wie Daten exportiert oder aufbewahrt werden. Beispielsweise kann die Stream-Größe in Kombination mit Ihrer Strategie bei vollen Einstellungen dazu führen, dass Daten gelöscht oder zurückgewiesen werden, bevor der Stream-Manager sie verarbeiten kann.
- Unterbrechen Sie Exporte und setzen Sie sie fort, z. B. wenn Exportaufgaben lange dauern und Sie Ihre Upload-Daten rationieren möchten.

Ihre Lambda-Funktionen folgen diesem allgemeinen Prozess, um einen Stream zu aktualisieren:

1. [Rufen Sie die Beschreibung des Streams ab.](#)
2. Aktualisieren Sie die Zieleigenschaften der entsprechenden `MessageStreamDefinition` und untergeordneten Objekte.

3. Übergeben Sie das Aktualisierte `MessageStreamDefinition`. Stellen Sie sicher, dass Sie die vollständigen Objektdefinitionen für den aktualisierten Stream angeben. Undefinierte Eigenschaften werden auf die Standardwerte zurückgesetzt.

Sie können die Sequenznummer der Nachricht angeben, die als Startnachricht im Export verwendet werden soll.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.11.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.6.0 | Java: 1.5.0 | Node.js: 1.7.0

Beispiele

Das folgende Snippet aktualisiert den Stream mit dem Namen `StreamName`. Es aktualisiert mehrere Eigenschaften eines Streams, der in Kinesis Data Streams exportiert wird.

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-Referenz: [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
Cloud.
                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-SDK-Referenz: [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
          Cloud.
          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.
  });

```

SDK-Referenz für Node.js: | [updateMessageStreamMessageStreamDefinition](#)

Einschränkungen für die Aktualisierung von Streams

Die folgenden Einschränkungen gelten für die Aktualisierung von Streams. Sofern in der folgenden Liste nicht anders angegeben, werden Aktualisierungen sofort wirksam.

- Sie können die Persistenz eines Streams nicht aktualisieren. Um dieses Verhalten zu ändern, [löschen Sie den Stream](#) und [erstellen Sie einen Stream](#), der die neue Persistenzrichtlinie definiert.
- Sie können die maximale Größe eines Streams nur unter den folgenden Bedingungen aktualisieren:
 - Die maximale Größe muss größer oder gleich der aktuellen Größe des Streams sein. Um diese Informationen zu finden, [beschreiben Sie den Stream](#) und überprüfen Sie dann den Speicherstatus des zurückgegebenen `MessageStreamInfo` Objekts.
 - Die maximale Größe muss größer oder gleich der Segmentgröße des Streams sein.
- Sie können die Stream-Segmentgröße auf einen Wert aktualisieren, der unter der maximalen Größe des Streams liegt. Die aktualisierte Einstellung gilt für neue Segmente.
- Aktualisierungen der Eigenschaft Time to Live (TTL) gelten für neue Anfügevorgänge. Wenn Sie diesen Wert verringern, löscht Stream Manager möglicherweise auch vorhandene Segmente, die die TTL überschreiten.
- Aktualisierungen der Strategie für die vollständige Eigenschaft gelten auch für neue Anfügevorgänge. Wenn Sie die Strategie so einrichten, dass die ältesten Daten überschrieben werden, überschreibt Stream Manager möglicherweise auch bestehende Segmente, die auf der neuen Einstellung basieren.
- Aktualisierungen der Eigenschaft „Flush on Write“ gelten für neue Nachrichten.
- Aktualisierungen der Exportkonfigurationen gelten für neue Exporte. Die Aktualisierungsanforderung muss alle Exportkonfigurationen enthalten, die Sie unterstützen möchten. Andernfalls löscht der Stream Manager sie.
 - Wenn Sie eine Exportkonfiguration aktualisieren, geben Sie den Bezeichner der Ziel-Exportkonfiguration an.
 - Um eine Exportkonfiguration hinzuzufügen, geben Sie eine eindeutige Kennung für die neue Exportkonfiguration an.
 - Um eine Exportkonfiguration zu löschen, lassen Sie die Exportkonfiguration weg.
- Um die Startsequenznummer einer Exportkonfiguration in einem Stream zu [aktualisieren](#), müssen Sie einen Wert angeben, der kleiner als die letzte Sequenznummer ist. Um diese Informationen zu finden, [beschreiben Sie den Stream](#) und überprüfen Sie dann den Speicherstatus des zurückgegebenen `MessageStreamInfo` Objekts.

Löschen eines Nachrichten-Streams

Löscht einen Stream. Wenn Sie einen Stream löschen, werden alle gespeicherten Daten für den Stream von der Festplatte gelöscht.

Voraussetzungen

Für diesen Vorgang gelten die folgenden Anforderungen:

- Minimale AWS IoT Greengrass Core-Version: 1.10.0
- Mindestversion des AWS IoT Greengrass Kern-SDK: Python: 1.5.0 | Java: 1.4.0 | Node.js: 1.6.0

Beispiele

Das folgende Snippet löscht den Stream mit dem Namen `StreamName`.

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python-SDK-Referenz: [deleteMessageStream](#)

Java

```
try (final StreamManagerClient client =
    GreengrassClientBuilder.streamManagerClient().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java-SDK-Referenz: [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.
});
```

SDK-Referenz für Node.js: [deleteMessageStream](#)

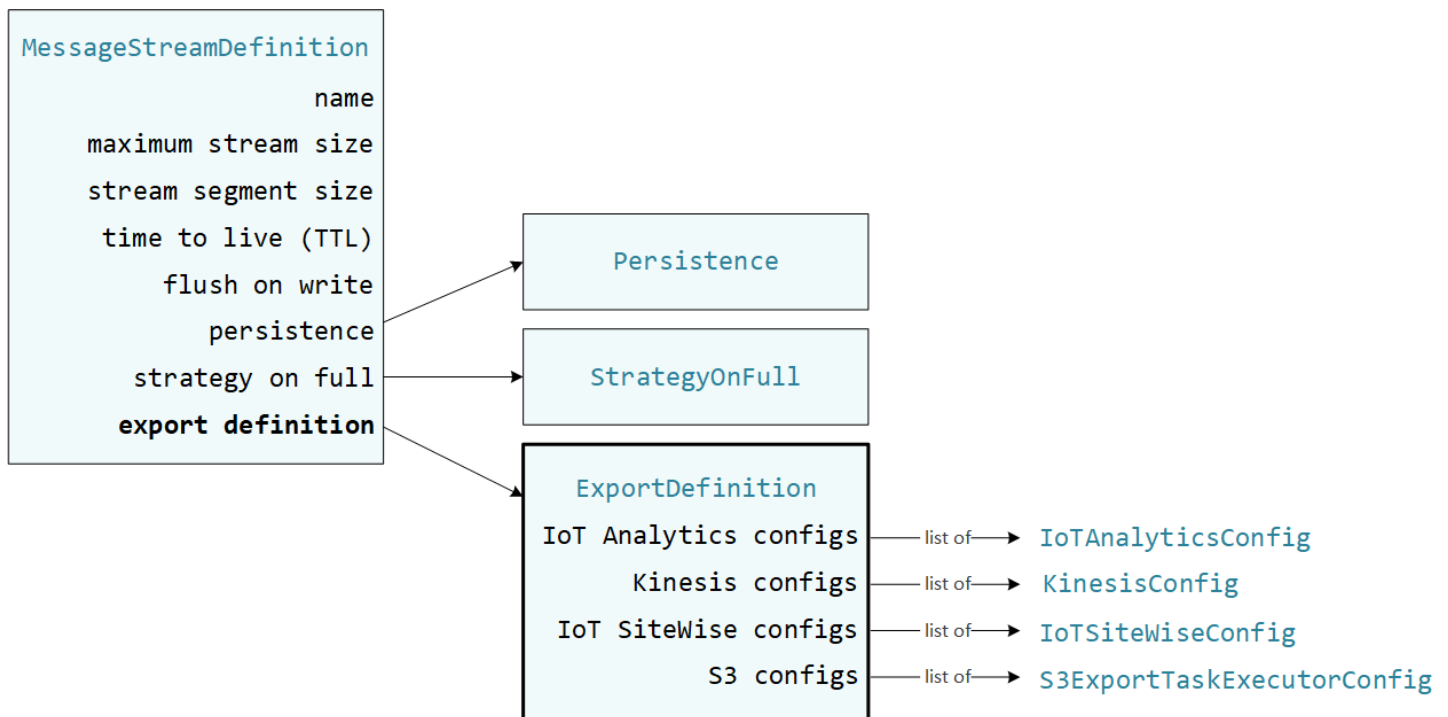
Weitere Informationen finden Sie auch unter

- [Verwalten von Daten-Streams](#)
- [the section called “Konfigurieren des Stream-Managers”](#)
- [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#)
- [the section called “Exportieren von Daten-Streams \(Konsole\)”](#)
- [the section called “Exportieren von Daten-Streams \(CLI\)”](#)
- StreamManagerClient in der AWS IoT Greengrass Core SDK-Referenz:
 - [Python](#)
 - [Java](#)
 - [Node.js](#)

Konfigurationen für unterstützte AWS Cloud Ziele exportieren

Benutzerdefinierte Lambda-Funktionen, die StreamManagerClient im AWS IoT Greengrass Core SDK verwendet werden, um mit dem Stream-Manager zu interagieren. Wenn eine Lambda-Funktion [einen Stream erstellt](#) oder [einen Stream aktualisiert](#), übergibt sie ein MessageStreamDefinition Objekt, das Stream-Eigenschaften darstellt, einschließlich der Exportdefinition. Das ExportDefinition Objekt enthält die für den Stream definierten

Exportkonfigurationen. Stream Manager verwendet diese Exportkonfigurationen, um zu bestimmen, wo und wie der Stream exportiert werden soll.



Sie können keine oder mehr Exportkonfigurationen für einen Stream definieren, einschließlich mehrerer Exportkonfigurationen für einen einzelnen Zieltyp. Sie können beispielsweise einen Stream in zwei AWS IoT Analytics Kanäle und einen Kinesis-Datenstream exportieren.

Bei fehlgeschlagenen Exportversuchen versucht Stream Manager kontinuierlich, Daten in Intervallen von bis zu fünf Minuten zu exportieren. AWS Cloud Für die Anzahl der Wiederholungsversuche gibt es keine Obergrenze.

Note

StreamManagerClient bietet auch ein Zielziel, mit dem Sie Streams auf einen HTTP-Server exportieren können. Dieses Ziel dient nur zu Testzwecken. Es ist weder stabil noch wird es für die Verwendung in Produktionsumgebungen unterstützt.

Unterstützte AWS Cloud Ziele

- [AWS IoT Analytics Kanäle](#)
- [Amazon Kinesis Kinesis-Datenströme](#)
- [AWS IoT SiteWise Eigenschaften von Vermögenswerten](#)

- [Amazon-S3-Objekte](#)

Sie sind für die Wartung dieser AWS Cloud Ressourcen verantwortlich.

AWS IoT Analytics Kanäle

Stream Manager unterstützt automatische Exporte nach AWS IoT Analytics. AWS IoT Analytics ermöglicht es Ihnen, erweiterte Analysen Ihrer Daten durchzuführen, um Geschäftsentscheidungen zu treffen und Modelle für maschinelles Lernen zu verbessern. Weitere Informationen finden Sie unter [Was ist AWS IoT Analytics?](#) im AWS IoT Analytics Benutzerhandbuch.

Im AWS IoT Greengrass Core SDK verwenden Ihre Lambda-Funktionen die, `IoTAnalyticsConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- [Io TAnalytics Config](#) im Python-SDK
- [Io TAnalytics Config](#) im Java SDK
- [Io TAnalytics Config](#) im SDK Node.js

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die Zielkanäle in AWS IoT Analytics müssen sich in derselben AWS-Konto und AWS-Region wie die Greengrass-Gruppe befinden.
- Sie [the section called "Greengrass-Gruppenrolle."](#) müssen die `iotanalytics:BatchPutMessage` Erlaubnis erteilen, Kanäle gezielt anzusprechen. Beispiel:

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"  
    ]  
}  
]  
}
```

Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Hinzufügen und Entfernen von IAM-Richtlinien](#).

Exportieren nach AWS IoT Analytics

Um einen Stream zu erstellen, in den exportiert wird AWS IoT Analytics, [erstellen Ihre Lambda-Funktionen einen Stream](#) mit einer Exportdefinition, die ein oder mehrere IoTAnalyticsConfig Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie den Zielkanal, die Batchgröße, das Batch-Intervall und die Priorität.

Wenn Ihre Lambda-Funktionen Daten von Geräten empfangen, [hängen sie Nachrichten](#), die einen Datenblock enthalten, an den Zielstream an.

Anschließend exportiert der Stream Manager die Daten auf der Grundlage der Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Amazon Kinesis Kinesis-Datenströme

Stream Manager unterstützt automatische Exporte nach Amazon Kinesis Data Streams. Kinesis Data Streams wird häufig verwendet, um große Datenmengen zu aggregieren und sie in ein Data Warehouse oder einen Map-Reduce-Cluster zu laden. Weitere Informationen finden Sie unter [Was ist Amazon Kinesis Data Streams?](#) im Amazon Kinesis Developer Guide.

Im AWS IoT Greengrass Core SDK verwenden Ihre Lambda-Funktionen die `KinesisConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- [KinesisConfig](#) im Python-SDK
- [KinesisConfig](#) im Java-SDK

- [KinesisConfig](#) SDK von Node.js

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Ziel-Streams in Kinesis Data Streams müssen sich in derselben AWS-Konto und AWS-Region wie die Greengrass-Gruppe befinden.
- Sie [the section called “Greengrass-Gruppenrolle.”](#) müssen die `kinesis:PutRecords` Erlaubnis erteilen, Datenstreams als Ziel zu verwenden. Beispiel:

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"
      ]
    }
  ]
}
```

Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Hinzufügen und Entfernen von IAM-Richtlinien](#).

In Kinesis Data Streams exportieren

Um einen Stream zu erstellen, der in Kinesis Data Streams exportiert wird, [erstellen Ihre Lambda-Funktionen einen Stream](#) mit einer Exportdefinition, die ein oder mehrere KinesisConfig Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie den Zieldatenstream, die Batchgröße, das Batch-Intervall und die Priorität.

Wenn Ihre Lambda-Funktionen Daten von Geräten empfangen, [hängen sie Nachrichten](#), die einen Datenblock enthalten, an den Zielstream an. Anschließend exportiert der Stream Manager die Daten auf der Grundlage der Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Stream Manager generiert eine eindeutige, zufällige UUID als Partitionsschlüssel für jeden Datensatz, der auf Amazon Kinesis hochgeladen wird.

AWS IoT SiteWise Eigenschaften von Vermögenswerten

Stream Manager unterstützt automatische Exporte nach AWS IoT SiteWise. AWS IoT SiteWise ermöglicht das Sammeln, Organisieren und Analysieren von Daten aus Industrieanlagen in großem Maßstab. Weitere Informationen finden Sie unter [Was ist AWS IoT SiteWise?](#) im AWS IoT SiteWise Benutzerhandbuch.

Im AWS IoT Greengrass Core SDK verwenden Ihre Lambda-Funktionen die, `IoTSiteWiseConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- [Io TSite WiseConfig](#) im Python-SDK
- [Io TSite WiseConfig](#) im Java-SDK
- [Io TSite WiseConfig](#) im SDK von Node.js

Note

AWS bietet auch die [the section called “IoT SiteWise”](#), bei der es sich um eine vorgefertigte Lösung handelt, die Sie mit OPC-UA-Quellen verwenden können.

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die Immobilien des AWS IoT SiteWise Zielobjekts in müssen sich in derselben AWS-Konto und AWS-Region wie die Greengrass-Gruppe befinden.

 Note

Eine Liste der Regionen, die AWS IoT SiteWise unterstützt werden, finden Sie in der AWS allgemeinen AWS IoT SiteWise [Referenz unter Endpunkte und Kontingente](#).

- Sie [the section called "Greengrass-Gruppenrolle."](#) müssen die `iotsitewise:BatchPutAssetPropertyValue` Erlaubnis erteilen, auf Objekteigenschaften zu zielen. In der folgenden Beispielrichtlinie wird der `iotsitewise:assetHierarchyPath` Bedingungsschlüssel verwendet, um Zugriff auf ein Zielstamm-Asset und dessen untergeordnete Objekte zu gewähren. Sie können das Condition aus der Richtlinie entfernen, um Zugriff auf alle Ihre AWS IoT SiteWise Ressourcen zu gewähren oder einzelne Ressourcen anzugeben 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/*"
          ]
        }
      }
    }
  ]
}
```

Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Hinzufügen und Entfernen von IAM-Richtlinien](#).

Wichtige Sicherheitsinformationen finden Sie im AWS IoT SiteWise Benutzerhandbuch unter [BatchPutAssetPropertyValue Autorisierung](#).

Exportieren nach AWS IoT SiteWise

Um einen Stream zu erstellen, in den exportiert wird AWS IoT SiteWise, [erstellen Ihre Lambda-Funktionen einen Stream](#) mit einer Exportdefinition, die ein oder mehrere `IoTSiteWiseConfig` Objekte enthält. Dieses Objekt definiert Exporteinstellungen wie Batchgröße, Batch-Intervall und Priorität.

Wenn Ihre Lambda-Funktionen Asset-Eigenschaftsdaten von Geräten empfangen, hängen sie Nachrichten, die die Daten enthalten, an den Zielstream an. Nachrichten sind JSON-serialisierte `PutAssetPropertyValueEntry` Objekte, die Eigenschaftswerte für eine oder mehrere Asset-Eigenschaften enthalten. Weitere Informationen finden Sie unter [Nachricht für Exportziele anhängen](#).
AWS IoT SiteWise

Note

Wenn Sie Daten an senden AWS IoT SiteWise, müssen Ihre Daten die Anforderungen der `BatchPutAssetPropertyValue` Aktion erfüllen. Weitere Informationen finden Sie unter [BatchPutAssetPropertyValue](#) in der AWS IoT SiteWise -API-Referenz.

Anschließend exportiert Stream Manager die Daten auf der Grundlage der Batch-Einstellungen und der Priorität, die in den Exportkonfigurationen des Streams definiert sind.

Sie können Ihre Stream-Manager-Einstellungen und die Lambda-Funktionslogik anpassen, um Ihre Exportstrategie zu entwerfen. Beispiel:

- Für Exporte fast in Echtzeit sollten Sie niedrige Einstellungen für Batchgröße und Intervalle festlegen und die Daten an den Stream anhängen, wenn sie empfangen werden.
- Um die Batchverarbeitung zu optimieren, Bandbreitenbeschränkungen zu verringern oder Kosten zu minimieren, können Ihre Lambda-Funktionen die für eine einzelne Anlageneigenschaft empfangenen timestamp-quality-value (TQV-) Datenpunkte bündeln, bevor sie an den Stream angehängt werden. Eine Strategie besteht darin, Einträge für bis zu 10 verschiedene Kombinationen von Eigenschaften und Vermögenswerten oder Eigenschaftsaliasnamen in einer Nachricht zu bündeln, anstatt mehr als einen Eintrag für dieselbe Immobilie zu senden. [Dies hilft dem Streammanager, die Kontingente einzuhalten.](#)
[AWS IoT SiteWise](#)

Amazon-S3-Objekte

Stream Manager unterstützt automatische Exporte nach Amazon S3. Sie können Amazon S3 verwenden, um große Datenmengen zu speichern und abzurufen. Weitere Informationen finden Sie unter [Was ist Amazon S3?](#) im Amazon Simple Storage Service Developer Guide.

Im AWS IoT Greengrass Core SDK verwenden Ihre Lambda-Funktionen die, `S3ExportTaskExecutorConfig` um die Exportkonfiguration für diesen Zieltyp zu definieren. Weitere Informationen finden Sie in der SDK-Referenz für Ihre Zielsprache:

- [S3 ExportTaskExecutorConfig](#) im Python-SDK
- [S3 ExportTaskExecutorConfig](#) im Java-SDK
- [S3 ExportTaskExecutorConfig](#) im SDK von Node.js

Voraussetzungen

Für dieses Exportziel gelten die folgenden Anforderungen:

- Die Amazon S3 S3-Ziel-Buckets müssen sich in derselben Gruppe AWS-Konto wie die Greengrass-Gruppe befinden.
- Wenn die [Standardcontainerisierung](#) für die Greengrass-Gruppe Greengrass-Container ist, müssen Sie den Parameter [STREAM_MANAGER_READ_ONLY_DIRS](#) so einstellen, dass ein Eingabedateiverzeichnis verwendet wird, das sich unter oder außerhalb des Root-Dateisystems befindet. `/tmp`
- Wenn eine Lambda-Funktion, die im Greengrass-Container-Modus ausgeführt wird, Eingabedateien in das Eingabedateiverzeichnis schreibt, müssen Sie eine lokale Volume-Ressource für das Verzeichnis erstellen und das Verzeichnis mit Schreibberechtigungen in den Container einbinden. Dadurch wird sichergestellt, dass die Dateien in das Root-Dateisystem geschrieben und außerhalb des Containers sichtbar sind. Weitere Informationen finden Sie unter [Zugreifen auf lokale Ressourcen](#).
- Sie [the section called "Greengrass-Gruppenrolle."](#) müssen die folgenden Berechtigungen für die Ziel-Buckets gewähren. Beispiel:

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/*"
  ]
}
```

Sie können detaillierten oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Hinzufügen und Entfernen von IAM-Richtlinien](#).

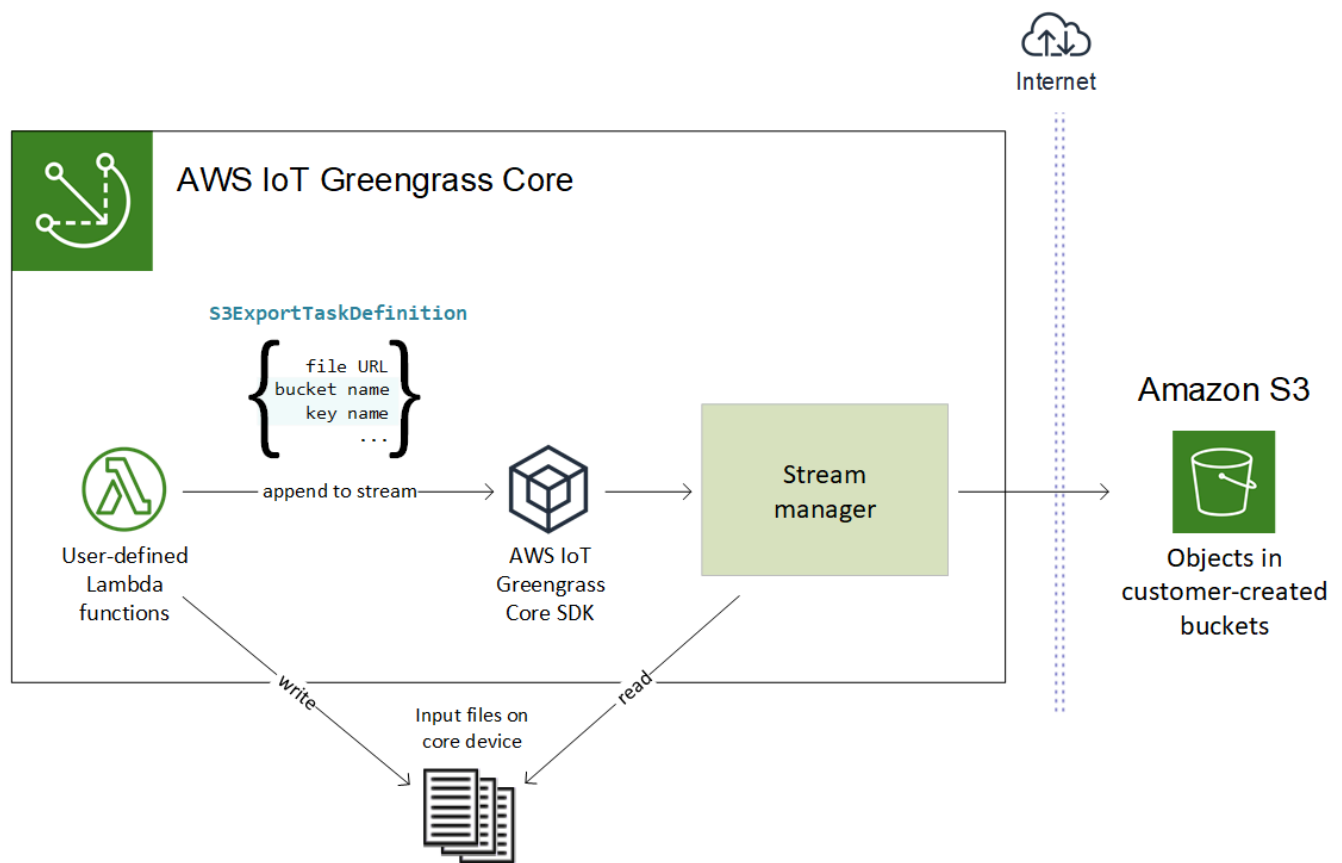
Exportieren nach Amazon S3

Um einen Stream zu erstellen, der nach Amazon S3 exportiert, verwenden Ihre Lambda-Funktionen das `S3ExportTaskExecutorConfig` Objekt, um die Exportrichtlinie zu konfigurieren. Die Richtlinie definiert Exporteinstellungen, wie z. B. den Schwellenwert und die Priorität für mehrteilige Uploads. Für Amazon S3 S3-Exporte lädt der Stream Manager Daten hoch, die er aus lokalen Dateien auf dem Kerngerät liest. Um einen Upload zu initiieren, hängen Ihre Lambda-Funktionen eine Exportaufgabe an den Zielstream an. Die Exportaufgabe enthält Informationen über die Eingabedatei und das Amazon S3 S3-Zielobjekt. Stream Manager führt Aufgaben in der Reihenfolge aus, in der sie an den Stream angehängt werden.

Note

Der Ziel-Bucket muss bereits in Ihrem vorhanden sein. AWS-Konto Wenn ein Objekt für den angegebenen Schlüssel nicht existiert, erstellt Stream Manager das Objekt für Sie.

Dieser allgemeine Arbeitsablauf ist in der folgenden Abbildung dargestellt.



Stream Manager verwendet die Eigenschaft „Schwellenwert für mehrteilige Uploads“, die Einstellung für die [Mindestgröße von Teilen](#) und die Größe der Eingabedatei, um zu bestimmen, wie Daten hochgeladen werden. Der Schwellenwert für mehrteilige Uploads muss größer oder gleich der Mindestgröße für Teile sein. Wenn Sie Daten parallel hochladen möchten, können Sie mehrere Streams erstellen.

Die Schlüssel, die Ihre Amazon S3 S3-Zielobjekte angeben, können gültige [DateTimeFormatterJava-Zeichenfolgen](#) in `!{timestamp: value}` Platzhaltern enthalten. Sie können diese Zeitstempel-Platzhalter verwenden, um Daten in Amazon S3 auf der Grundlage der Uhrzeit zu partitionieren, zu der die Eingabedateidaten hochgeladen wurden. Der folgende Schlüsselname wird beispielsweise in einen Wert wie aufgelöst. `my-key/2020/12/31/data.txt`

```
my-key/!{timestamp:YYYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

Note

Wenn Sie den Exportstatus für einen Stream überwachen möchten, erstellen Sie zuerst einen Status-Stream und konfigurieren Sie dann den Exportstream so, dass er ihn

verwendet. Weitere Informationen finden Sie unter [the section called “Überwachen Sie die Exportaufgaben”](#).

Eingabedaten verwalten

Sie können Code verfassen, den IoT-Anwendungen verwenden, um den Lebenszyklus der Eingabedaten zu verwalten. Der folgende Beispiel-Workflow zeigt, wie Sie Lambda-Funktionen verwenden können, um diese Daten zu verwalten.

1. Ein lokaler Prozess empfängt Daten von Geräten oder Peripheriegeräten und schreibt die Daten dann in Dateien in einem Verzeichnis auf dem Kerngerät. Dies sind die Eingabedateien für den Stream-Manager.

Note

Informationen darüber, ob Sie den Zugriff auf das Eingabedateiverzeichnis konfigurieren müssen, finden Sie im Parameter [STREAM_MANAGER_READ_ONLY_DIRS](#). [Der Prozess, in dem Stream Manager ausgeführt wird, erbt alle Dateisystemberechtigungen der Standardzugriffsidentität für die Gruppe](#). Der Stream-Manager muss über die Zugriffsberechtigung für die Eingabedateien verfügen. Sie können den `chmod(1)` Befehl verwenden, um die Berechtigungen der Dateien bei Bedarf zu ändern.

2. Eine Lambda-Funktion scannt das Verzeichnis und [hängt eine Exportaufgabe](#) an den Zielstream an, wenn eine neue Datei erstellt wird. Die Aufgabe ist ein JSON-serialisiertes `S3ExportTaskDefinition` Objekt, das die URL der Eingabedatei, den Amazon S3 S3-Ziel-Bucket und -Schlüssel sowie optionale Benutzermetadaten angibt.
3. Stream Manager liest die Eingabedatei und exportiert die Daten in der Reihenfolge der angehängten Aufgaben nach Amazon S3. Der Ziel-Bucket muss bereits in Ihrem AWS-Konto vorhanden sein. Wenn ein Objekt für den angegebenen Schlüssel nicht existiert, erstellt Stream Manager das Objekt für Sie.
4. Die Lambda-Funktion [liest Nachrichten](#) aus einem Status-Stream, um den Exportstatus zu überwachen. Nach Abschluss der Exportaufgaben kann die Lambda-Funktion die entsprechenden Eingabedateien löschen. Weitere Informationen finden Sie unter [the section called “Überwachen Sie die Exportaufgaben”](#).

Überwachen Sie die Exportaufgaben

Sie können Code verfassen, den IoT-Anwendungen verwenden, um den Status Ihrer Amazon S3 S3-Exporte zu überwachen. Ihre Lambda-Funktionen müssen einen Status-Stream erstellen und dann den Export-Stream so konfigurieren, dass Statusaktualisierungen in den Status-Stream geschrieben werden. Ein einziger Status-Stream kann Statusaktualisierungen von mehreren Streams empfangen, die nach Amazon S3 exportiert werden.

[Erstellen Sie zunächst einen Stream](#), der als Status-Stream verwendet werden soll. Sie können die Größe und die Aufbewahrungsrichtlinien für den Stream konfigurieren, um die Lebensdauer der Statusmeldungen zu steuern. Beispiel:

- Stellen Persistence Sie Memory diese Option ein, wenn Sie die Statusmeldungen nicht speichern möchten.
- Auf StrategyOnFull einstellen, OverwriteOldestData damit neue Statusmeldungen nicht verloren gehen.

Erstellen oder aktualisieren Sie dann den Exportstream, um den Status-Stream zu verwenden. Legen Sie insbesondere die Statuskonfigurationseigenschaft der S3ExportTaskExecutorConfig Exportkonfiguration des Streams fest. Dadurch wird der Stream-Manager angewiesen, Statusmeldungen über die Exportaufgaben in den Status-Stream zu schreiben. Geben Sie im StatusConfig Objekt den Namen des Status-Streams und den Grad der Ausführlichkeit an. Die folgenden unterstützten Werte reichen von der geringsten Ausführlichkeit (ERROR) bis zur höchsten Ausführlichkeit (). TRACE Der Standardwert ist INFO.

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

Der folgende Beispiel-Workflow zeigt, wie Lambda-Funktionen einen Status-Stream verwenden könnten, um den Exportstatus zu überwachen.

1. Wie im vorherigen Workflow beschrieben, [hängt eine Lambda-Funktion eine Exportaufgabe an einen](#) Stream an, der so konfiguriert ist, dass Statusmeldungen über Exportaufgaben in einen Status-Stream geschrieben werden. Die Anfügeoperation gibt eine Sequenznummer zurück, die die Aufgaben-ID darstellt.
2. Eine Lambda-Funktion [liest Nachrichten](#) sequentiell aus dem Status-Stream und filtert die Nachrichten dann basierend auf dem Streamnamen und der Task-ID oder basierend auf einer Exportaufgabeneigenschaft aus dem Nachrichtenkontext. Die Lambda-Funktion kann beispielsweise nach der Eingabedatei-URL der Exportaufgabe filtern, die durch das `S3ExportTaskDefinition` Objekt im Nachrichtenkontext dargestellt wird.

Die folgenden Statuscodes geben an, dass eine Exportaufgabe den Status „Abgeschlossen“ erreicht hat:

- **Success.** Der Upload wurde erfolgreich abgeschlossen.
- **Failure.** Stream Manager ist auf einen Fehler gestoßen, z. B. ist der angegebene Bucket nicht vorhanden. Nachdem Sie das Problem behoben haben, können Sie die Exportaufgabe erneut an den Stream anhängen.
- **Canceled.** Die Aufgabe wurde abgebrochen, weil die Stream- oder Exportdefinition gelöscht wurde oder der Zeitraum *time-to-live* (TTL) der Aufgabe abgelaufen ist.

Note

Die Aufgabe hat möglicherweise auch den Status `InProgress`. Wenn Stream Manager Warnungen ausgibt, wenn ein Ereignis einen Fehler zurückgibt, der die Ausführung der Aufgabe nicht beeinträchtigt. Wenn beispielsweise ein abgebrochener teilweiser Upload nicht bereinigt werden kann, wird eine Warnung zurückgegeben.

3. Nach Abschluss der Exportaufgaben kann die Lambda-Funktion die entsprechenden Eingabedateien löschen.

Das folgende Beispiel zeigt, wie eine Lambda-Funktion Statusmeldungen lesen und verarbeiten könnte.

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
```

```

        Status,
        StatusConfig,
        StatusLevel,
        StatusMessage,
        StreamManagerClient,
    )
from greengrasssdk.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-Referenz: [read_messages](#) | [StatusMessage](#)

Java

```

import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
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 =
    GreengrassClientBuilder.streamManagerClient().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 SDK-Referenz: [readMessages](#) | [StatusMessage](#)

Node.js

```

const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

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.

```

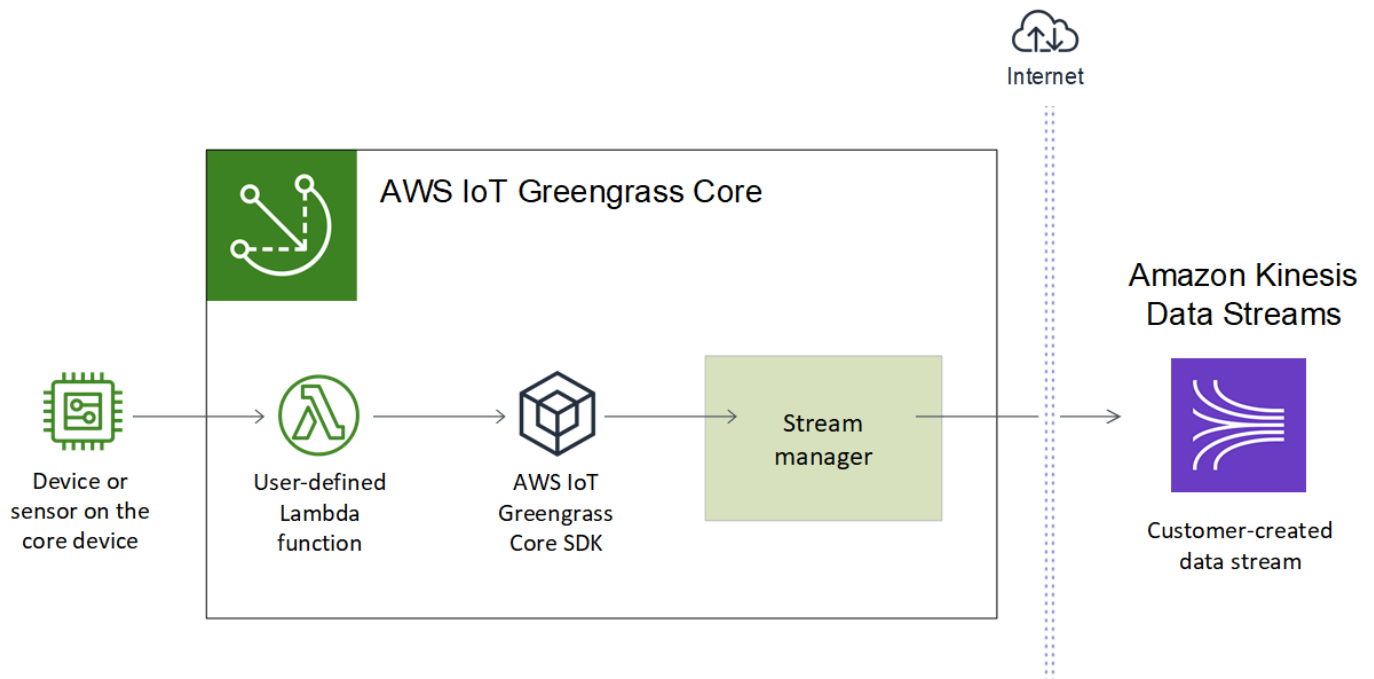
```
});
```

Node.js SDK-Referenz: [readMessages](#) | [StatusMessage](#)

Datenströme in die AWS Cloud (Konsole) exportieren

Dieses Tutorial zeigt Ihnen, wie Sie die AWS IoT Konsole verwenden, um eine AWS IoT Greengrass Gruppe mit aktiviertem Stream-Manager zu konfigurieren und bereitzustellen. Die Gruppe enthält eine benutzerdefinierte Lambda-Funktion, die in einen Stream im Stream-Manager schreibt, der dann automatisch in den exportiert wird. AWS Cloud

Stream Manager macht die Aufnahme, Verarbeitung und den Export von Datenströmen mit hohem Volumen effizienter und zuverlässiger. In diesem Tutorial erstellen Sie eine `TransferStream` Lambda-Funktion, die IoT-Daten verbraucht. Die Lambda-Funktion verwendet das AWS IoT Greengrass Core SDK, um einen Stream im Stream Manager zu erstellen und dann in ihn zu lesen und zu schreiben. Stream Manager exportiert den Stream dann nach Kinesis Data Streams. Im folgenden Diagramm wird dieser Workflow veranschaulicht.



Der Schwerpunkt dieses Tutorials liegt darauf, zu zeigen, wie benutzerdefinierte Lambda-Funktionen das `StreamManagerClient` Objekt im AWS IoT Greengrass Core SDK verwenden, um mit dem Stream Manager zu interagieren. Der Einfachheit halber generiert die Python-Lambda-Funktion, die Sie für dieses Tutorial erstellen, simulierte Gerätedaten.

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Kern (v1.10 oder höher). Informationen zum Erstellen einer Greengrass-Gruppe und eines Greengrass-Kerns finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#). Das Tutorial „Erste Schritte“ enthält auch Schritte zur Installation der AWS IoT Greengrass Core-Software.

Note

Stream Manager wird auf OpenWrt Distributionen nicht unterstützt.

- Die Java 8-Laufzeitumgebung (JDK 8) muss auf dem Core-Gerät installiert sein.
 - Führen Sie für Debian-basierte Distributionen (einschließlich Raspbian) oder Ubuntu-basierte Distributionen den folgenden Befehl aus:

```
sudo apt install openjdk-8-jdk
```

- Führen Sie für Red Hat-basierte Distributionen (einschließlich Amazon Linux) den folgenden Befehl aus:

```
sudo yum install java-1.8.0-openjdk
```

Weitere Informationen finden Sie unter [How to download and install prebuilt OpenJDK packages](#) in der OpenJDK-Dokumentation.

- AWS IoT Greengrass Core SDK für Python v1.5.0 oder höher. Um es `StreamManagerClient` im AWS IoT Greengrass Core SDK für Python zu verwenden, müssen Sie:
 - Installieren Sie Python 3.7 oder höher auf dem Core-Gerät.
 - Nehmen Sie das SDK und seine Abhängigkeiten in Ihr Lambda-Funktionsbereitstellungspaket auf. Anweisungen finden Sie in diesem Tutorial.

Tip

Sie können `StreamManagerClient` mit Java oder NodeJS verwenden. Beispielcode finden Sie unter dem [AWS IoT Greengrass Core SDK for Java](#) und dem [AWS IoT Greengrass Core SDK für Node.js](#) GitHub.

- Ein Zielstream mit dem Namen, der in Amazon Kinesis Data Streams in derselben AWS-Region Greengrass-Gruppe **MyKinesisStream** erstellt wurde. Weitere Informationen finden Sie unter [Create a Stream](#) im Amazon Kinesis Developer Guide.

 Note

In diesem Tutorial exportiert Stream Manager Daten in Kinesis Data Streams, was zu Gebühren für Sie AWS-Konto führt. Preisinformationen finden Sie unter Preise für [Kinesis Data Streams](#).

Um Kosten zu vermeiden, können Sie dieses Lernprogramm ausführen, ohne einen Kinesis-Datenstream zu erstellen. In diesem Fall überprüfen Sie die Protokolle, um festzustellen, dass der Stream-Manager versucht hat, den Stream nach Kinesis Data Streams zu exportieren.

- Dem wurde eine IAM-Richtlinie hinzugefügt [the section called “Greengrass-Gruppenrolle.”](#), die die `kinesis:PutRecords` Aktion für den Zieldatenstream ermöglicht, wie im folgenden Beispiel gezeigt:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:123456789012:stream/MyKinesisStream"
      ]
    }
  ]
}
```

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#)
2. [Erstellen einer Lambda-Funktion](#)

3. [Hinzufügen einer Funktion zur Gruppe](#)
4. [Aktivieren des Stream-Managers](#)
5. [Konfigurieren der lokalen Protokollierung](#)
6. [Bereitstellen der Gruppe](#)
7. [Testen der Anwendung](#)

Für dieses Tutorial benötigen Sie ungefähr 20 Minuten.

Schritt 1: Erstellen Sie ein Lambda-Funktions-Bereitstellungspaket

In diesem Schritt erstellen Sie ein Paket zur Bereitstellung von Lambda-Funktionen, das Python-Funktionscode und Abhängigkeiten enthält. Sie laden dieses Paket später hoch, wenn Sie die Lambda-Funktion in AWS Lambda erstellen. Die Lambda-Funktion verwendet das AWS IoT Greengrass Core SDK, um lokale Streams zu erstellen und mit ihnen zu interagieren.

Note

Ihre benutzerdefinierten Lambda-Funktionen müssen das [AWS IoT Greengrass Core SDK](#) verwenden, um mit dem Stream Manager zu interagieren. Weitere Informationen zu den Anforderungen für den Greengrass Stream-Manager finden Sie im Artikel über die [Voraussetzungen für Greengrass Stream-Manager](#).

1. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) v1.5.0 oder höher herunter.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der `greengrasssdk`-Ordner.
3. Installieren Sie Paketabhängigkeiten, die mit dem SDK in Ihr Lambda-Funktionsbereitstellungspaket aufgenommen werden sollen.
 1. Navigieren Sie zum SDK-Verzeichnis, das die `requirements.txt`-Datei enthält. Diese Datei listet die Abhängigkeiten auf.
 2. Installieren Sie die SDK-Abhängigkeiten. Führen Sie beispielsweise den folgenden `pip`-Befehl aus, um sie im aktuellen Verzeichnis zu installieren:

```
pip install --target . -r requirements.txt
```

4. Speichern Sie die folgende Pythoncode-Funktion in einer lokalen Datei namens `transfer_stream.py`.

 Tip

Beispielcode, der Java und NodeJS verwendet, finden Sie im [AWS IoT Greengrass Core SDK for Java](#) und [AWS IoT Greengrass Core SDK for Node.js](#) unter. GitHub

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)

# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
# exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
# MB).
# Any data appended after the stream reaches the size limit continues to be
# appended, and
# stream manager deletes the oldest data until the total stream size is back under
# 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
# script is
# uploaded to Kinesis and you will be charged for that usage.
```

```
def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHJKLMNOP".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
```

```
# Now start putting in random data between 0 and 1000 to emulate device
sensor input
while True:
    logger.debug("Appending new random integer to stream")
    client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
    time.sleep(1)

except asyncio.TimeoutError:
    logger.exception("Timed out while executing")
except Exception:
    logger.exception("Exception while running")

def function_handler(event, context):
    return

logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Packen Sie die folgenden Elemente in einer ZIP-Datei mit dem Namen `transfer_stream_python.zip`. Dies ist Ihr Lambda-Funktionsbereitstellungspaket.

- `transfer_stream.py`. App-Logik.
- `greengrasssdk`. Erforderliche Bibliothek für Python Greengrass Lambda-Funktionen, die MQTT-Nachrichten veröffentlichen.

[Stream Manager-Operationen](#) sind in Version 1.5.0 oder höher des AWS IoT Greengrass Core SDK für Python verfügbar.

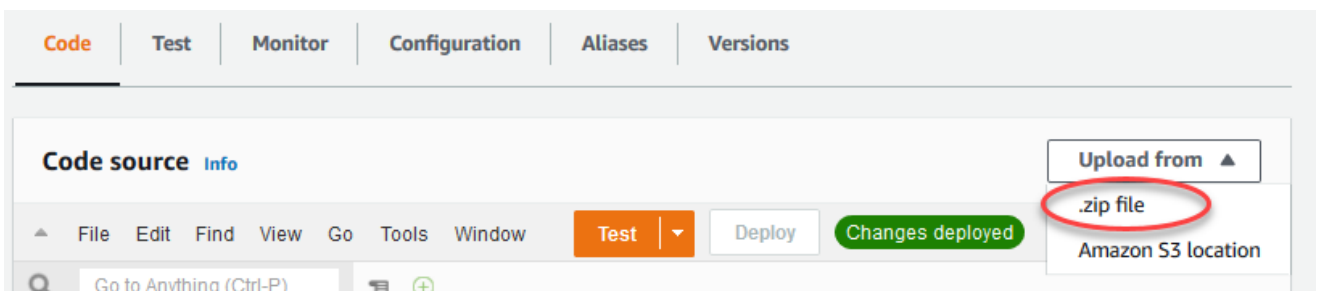
- Die Abhängigkeiten, die Sie für das AWS IoT Greengrass Core SDK für Python installiert haben (z. B. die `cbor2` Verzeichnisse).

Wenn Sie die `zip`-Datei erstellen, schließen Sie nur diese Elemente ein, nicht den enthaltenen Ordner.

Schritt 2: Erstellen einer Lambda-Funktion

In diesem Schritt verwenden Sie die AWS Lambda Konsole, um eine Lambda-Funktion zu erstellen und sie für die Verwendung Ihres Bereitstellungspakets zu konfigurieren. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

1. Erstellen Sie zunächst die Lambda-Funktion.
 - a. Wählen Sie in der AWS-Managementkonsole Dienste aus und öffnen Sie die AWS Lambda Konsole.
 - b. Wählen Sie „Funktion erstellen“ und anschließend „Von Grund auf neu erstellen“.
 - c. Verwenden Sie im Abschnitt Basic information (Basisinformationen) folgende Werte:
 - Geben Sie für Function name (Funktionsname) **TransferStream** ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.
 - d. Klicken Sie unten auf der Seite auf Create function.
2. Registrieren Sie als Nächstes den Handler und laden Sie Ihr Lambda-Funktionsbereitstellungspaket hoch.
 - a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Upload von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Hochladen und dann Ihr `transfer_stream_python.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.

- Geben Sie unter Handler **transfer_stream.function_handler** ein.
- d. Wählen Sie Speichern.

 Note

Die Testtaste auf der AWS Lambda Konsole funktioniert mit dieser Funktion nicht. Das AWS IoT Greengrass Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig in der AWS Lambda Konsole auszuführen. Diese Module (zum Beispiel `greengrass_common`) werden den Funktionen zur Verfügung gestellt, nachdem sie auf Ihrem Greengrass-Kern bereitgestellt wurden.

3. Veröffentlichen Sie jetzt die erste Version Ihrer Lambda-Funktion und erstellen Sie einen [Alias für die Version](#).

 Note

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

- a. Wählen Sie im Menü Actions die Option Publish new version aus.
- b. Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.
- c. Wählen Sie auf der Konfigurationsseite TransferStream: 1 im Menü Aktionen die Option Alias erstellen aus.
- d. Geben Sie auf der Seite Create a new alias folgende Werte an:
- Geben Sie unter Name **GG_TransferStream** ein.
 - Wählen Sie für Version die Option 1.

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

- e. Wählen Sie Erstellen aus.

Jetzt sind Sie bereit, die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzuzufügen.

Schritt 3: Fügen Sie der Greengrass-Gruppe eine Lambda-Funktion hinzu

In diesem Schritt fügen Sie der Gruppe die Lambda-Funktion hinzu und konfigurieren dann ihre Lebenszyklus- und Umgebungsvariablen. Weitere Informationen finden Sie unter [the section called "Steuerung der Ausführung von Greengrass Lambda-Funktionen"](#).

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
4. Wählen Sie unter Meine Lambda-Funktionen die Option Hinzufügen aus.
5. Wählen Sie auf der Seite Lambda-Funktion hinzufügen die Lambda-Funktion für Ihre Lambda-Funktion aus.
6. Wählen Sie für die Lambda-Version TransferStream Alias:GG_.

Konfigurieren Sie nun Eigenschaften, die das Verhalten der Lambda-Funktion in der Greengrass-Gruppe bestimmen.

7. Nehmen Sie im Abschnitt Lambda-Funktionskonfiguration die folgenden Änderungen vor:
 - Legen Sie die Speichergrenze auf 32 MB fest.
 - Wählen Sie für Pinned die Option True aus.

 Note

Eine langlebige (oder fixierte) Lambda-Funktion wird nach dem Start automatisch AWS IoT Greengrass gestartet und läuft in ihrem eigenen Container weiter. Dies steht im Gegensatz zu einer On-Demand-Lambda-Funktion, die startet, wenn sie aufgerufen wird,

und stoppt, wenn keine Aufgaben mehr zur Ausführung übrig sind. Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

8. Wählen Sie Lambda-Funktion hinzufügen.

Schritt 4: Aktivieren des Stream-Managers

In diesem Schritt stellen Sie sicher, dass der Stream-Manager aktiviert ist.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
2. Wählen Sie unter System Lambda functions die Option Stream Manager aus und überprüfen Sie den Status. Wenn diese Option deaktiviert ist, wählen Sie Bearbeiten aus. Wählen Sie dann Aktivieren und Speichern aus. Sie können die Standardparametereinstellungen für dieses Tutorial verwenden. Weitere Informationen finden Sie unter [the section called “Konfigurieren des Stream-Managers”](#).

Note

Wenn Sie die Konsole verwenden, um Stream Manager zu aktivieren und die Gruppe bereitzustellen, ist die Speichergröße für Stream Manager standardmäßig auf 4194304 KB (4 GB) festgelegt. Wir empfehlen, die Speichergröße auf mindestens 128000 KB festzulegen.

Schritt 5: Konfigurieren der lokalen Protokollierung

In diesem Schritt konfigurieren Sie AWS IoT Greengrass Systemkomponenten, benutzerdefinierte Lambda-Funktionen und Konnektoren in der Gruppe, um Protokolle in das Dateisystem des Kerngeräts zu schreiben. Sie können Protokolle verwenden, um Probleme zu beheben, die auftreten können. Weitere Informationen finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

1. Überprüfen Sie unter Konfiguration der lokalen Protokolle, ob die lokale Protokollierung konfiguriert ist.
2. Wenn Logs nicht für Greengrass-Systemkomponenten oder benutzerdefinierte Lambda-Funktionen konfiguriert sind, wählen Sie Bearbeiten.

3. Wählen Sie die Protokollebene der Benutzerlambda-Funktionen und die Protokollebene des Greengrass-Systems aus.
4. Behalten Sie die Standardwerte für die Protokollierungsebene und Kontingentsgrenze bei. Wählen Sie anschließend Save (Speichern) aus.

Schritt 6: Bereitstellen der Greengrass-Gruppe

Stellen Sie die Gruppe auf dem Core-Gerät bereit.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/ggc-version/bin/daemon` enthält, dann wird der Daemon ausgeführt.

Note

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Deploy aus.
3.
 - a. Wählen Sie auf der Registerkarte Lambda-Funktionen im Abschnitt System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
 - b. Wählen Sie im Dialogfeld IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben aus.
 - c. Wählen Sie Speichern.

Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Eine automatische Erkennung wird empfohlen,

unterstützt aber AWS IoT Greengrass auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

 Note

Wenn Sie dazu aufgefordert werden, erteilen Sie die Erlaubnis, die [Greengrass-Servicerolle zu erstellen und sie Ihrer AWS-Konto aktuellen AWS-Region Rolle](#) zuzuordnen. Diese Rolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Diensten.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss sollte der für die Bereitstellung angezeigte Status Abgeschlossen lauten.

Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Schritt 7: Testen der Anwendung

Die `TransferStream` Lambda-Funktion generiert simulierte Gerätedaten. Sie schreibt Daten in einen Stream, den der Stream-Manager in den Kinesis-Zielfdatenstream exportiert.

1. Wählen Sie in der Amazon Kinesis Kinesis-Konsole unter Kinesis-Datenstreams die Option. `MyKinesisStream`

 Note

Wenn Sie das Lernprogramm ohne Kinesis-Zielfdatenstream ausgeführt haben, [suchen Sie in der Protokolldatei](#) nach dem Stream-Manager (`GGStreamManager`). Wenn `export stream MyKinesisStream doesn't exist` in einer Fehlermeldung enthalten ist, ist der Test erfolgreich. Dieser Fehler bedeutet, dass der Service versucht hat, in den Stream zu exportieren, der Stream jedoch nicht existiert.

2. Wählen Sie auf der `MyKinesisStream`Seite Monitoring aus. Wenn der Test erfolgreich ist, sollten Sie die Daten in den Put Records (Datensätze übergeben)-Diagrammen sehen. Je nach Verbindung kann es eine Minute dauern, bis die Daten angezeigt werden.

⚠ Important

Löschen Sie nach Abschluss des Tests den Kinesis-Datenstream, um zusätzliche Kosten zu vermeiden.

Oder führen Sie die folgenden Befehle aus, um den Greengrass-Daemon zu stoppen. Dadurch wird verhindert, dass der Core Nachrichten sendet, bis Sie bereit sind, den Test fortzusetzen.

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

3. Entfernen Sie die TransferStreamLambda-Funktion aus dem Kern.
 - a. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
 - b. Wählen Sie unter Greengrass-Gruppen Ihre Gruppe aus.
 - c. Wählen Sie auf der Lambdas-Seite die Ellipsen (...) für die TransferStreamFunktion aus, und wählen Sie dann Funktion entfernen aus.
 - d. Wählen Sie unter Actions (Aktionen) die Option Deploy (Bereitstellen) aus.

Um Protokollierungsinformationen anzuzeigen oder Probleme mit Streams zu beheben, suchen Sie in den Protokollen nach den Funktionen TransferStream und GGStreamManager. Sie müssen über root Berechtigungen zum Lesen von AWS IoT Greengrass Protokollen im Dateisystem verfügen.

- TransferStream schreibt Protokolleinträge in *greengrass-root*/ggc/var/log/user/*region*/*account-id*/TransferStream.log.
- GGStreamManager schreibt Protokolleinträge in *greengrass-root*/ggc/var/log/system/GGStreamManager.log.

Wenn Sie weitere Informationen zur Fehlerbehebung benötigen, können Sie [die Protokollierungsebene für Benutzer-Lambda-Protokolle auf Debug-Protokolle festlegen](#) und die Gruppe dann erneut bereitstellen.

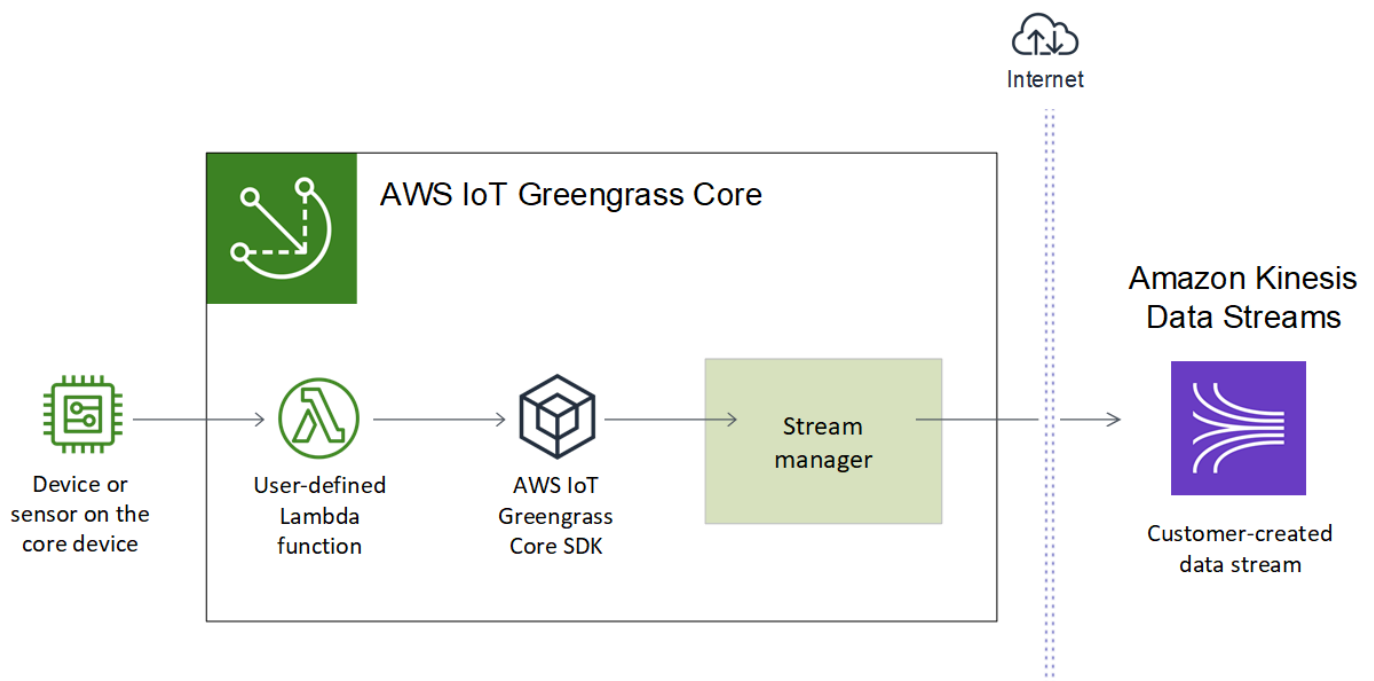
Weitere Informationen finden Sie auch unter

- [Verwalten von Daten-Streams](#)
- [the section called “Konfigurieren des Stream-Managers”](#)
- [the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”](#)
- [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#)
- [the section called “Exportieren von Daten-Streams \(CLI\)”](#)

Datenströme in die AWS Cloud (CLI) exportieren

In diesem Tutorial erfahren Sie, wie Sie mit AWS CLI dem eine AWS IoT Greengrass Gruppe mit aktiviertem Stream-Manager konfigurieren und bereitstellen. Die Gruppe enthält eine benutzerdefinierte Lambda-Funktion, die in einen Stream im Stream-Manager schreibt, der dann automatisch in den exportiert wird. AWS Cloud

Stream Manager macht die Aufnahme, Verarbeitung und den Export von Datenströmen mit hohem Volumen effizienter und zuverlässiger. In diesem Tutorial erstellen Sie eine TransferStream Lambda-Funktion, die IoT-Daten verbraucht. Die Lambda-Funktion verwendet das AWS IoT Greengrass Core SDK, um einen Stream im Stream Manager zu erstellen und dann in ihn zu lesen und zu schreiben. Stream Manager exportiert den Stream dann nach Kinesis Data Streams. Im folgenden Diagramm wird dieser Workflow veranschaulicht.



Der Schwerpunkt dieses Tutorials liegt darauf, zu zeigen, wie benutzerdefinierte Lambda-Funktionen das `StreamManagerClient` Objekt im AWS IoT Greengrass Core SDK verwenden, um mit dem Stream Manager zu interagieren. Der Einfachheit halber generiert die Python-Lambda-Funktion, die Sie für dieses Tutorial erstellen, simulierte Gerätedaten.

Wenn Sie die AWS IoT Greengrass API verwenden, die die Greengrass-Befehle in der enthält AWS CLI, um eine Gruppe zu erstellen, ist der Stream-Manager standardmäßig deaktiviert. Um den Stream Manager auf Ihrem Core zu aktivieren, [erstellen Sie eine Funktionsdefinitionsversion](#), die die `GGStreamManager` Lambda-Systemfunktion enthält, und eine Gruppenversion, die auf die neue Funktionsdefinitionsversion verweist. Anschließend stellen Sie die Gruppe bereit.

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Kern (v1.10 oder höher). Informationen zum Erstellen einer Greengrass-Gruppe und eines Greengrass-Kerns finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#). Das Tutorial „Erste Schritte“ enthält auch Schritte zur Installation der AWS IoT Greengrass Core-Software.

Note

Stream Manager wird auf OpenWrt Distributionen nicht unterstützt.

- Die Java 8-Laufzeitumgebung (JDK 8) muss auf dem Core-Gerät installiert sein.
- Führen Sie für Debian-basierte Distributionen (einschließlich Raspbian) oder Ubuntu-basierte Distributionen den folgenden Befehl aus:

```
sudo apt install openjdk-8-jdk
```

- Führen Sie für Red Hat-basierte Distributionen (einschließlich Amazon Linux) den folgenden Befehl aus:

```
sudo yum install java-1.8.0-openjdk
```

Weitere Informationen finden Sie unter [How to download and install prebuilt OpenJDK packages](#) in der OpenJDK-Dokumentation.

- AWS IoT Greengrass Core SDK für Python v1.5.0 oder höher. Um es `StreamManagerClient` im AWS IoT Greengrass Core SDK für Python zu verwenden, müssen Sie:
 - Installieren Sie Python 3.7 oder höher auf dem Core-Gerät.
 - Nehmen Sie das SDK und seine Abhängigkeiten in Ihr Lambda-Funktionsbereitstellungspaket auf. Anweisungen finden Sie in diesem Tutorial.

i Tip

Sie können `StreamManagerClient` mit Java oder NodeJS verwenden. Beispielcode finden Sie unter [AWS IoT Greengrass Core SDK for Java](#) und [AWS IoT Greengrass Core SDK für Node.js](#) GitHub.

- Ein Zielstream mit dem Namen, der in Amazon Kinesis Data Streams in derselben AWS-Region Greengrass-Gruppe **MyKinesisStream** erstellt wurde. Weitere Informationen finden Sie unter [Create a Stream](#) im Amazon Kinesis Developer Guide.

i Note

In diesem Tutorial exportiert Stream Manager Daten in Kinesis Data Streams, was zu Gebühren für Sie AWS-Konto führt. Preisinformationen finden Sie unter Preise für [Kinesis Data Streams](#).

Um Kosten zu vermeiden, können Sie dieses Lernprogramm ausführen, ohne einen Kinesis-Datenstream zu erstellen. In diesem Fall überprüfen Sie die Protokolle, um festzustellen, dass der Stream-Manager versucht hat, den Stream nach Kinesis Data Streams zu exportieren.

- Dem wurde eine IAM-Richtlinie hinzugefügt [the section called “Greengrass-Gruppenrolle.”](#), die die `kinesis:PutRecords` Aktion für den Zieldatenstream ermöglicht, wie im folgenden Beispiel gezeigt:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ]
    }
  ]
}
```

```
    ],  
    "Resource": [  
        "arn:aws:kinesis:us-east-1:123456789012:stream/MyKinesisStream"  
    ]  
  }  
]  
}
```

- Das AWS CLI ist auf Ihrem Computer installiert und konfiguriert. Weitere Informationen finden Sie [unter Installation AWS Command Line Interface](#) und [Konfiguration von AWS CLI im AWS Command Line Interface](#) Benutzerhandbuch.

Die Beispielbefehle in diesem Tutorial sind für Linux und andere Unix-basierte Systeme geschrieben. Wenn Sie Windows verwenden, finden Sie unter [Angaben von Parameterwerten für die AWS Befehlszeilenschnittstelle](#) weitere Informationen zu Unterschieden in der Syntax.

Wenn der Befehl eine JSON-Zeichenkette enthält, zeigt das Tutorial ein Beispiel, das das JSON auf einer einzigen Zeile hat. Auf einigen Systemen ist es möglicherweise effizienter, Befehle in diesem Format zu bearbeiten und auszuführen.

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#)
2. [Erstellen einer Lambda-Funktion](#)
3. [Erstellen einer Funktionsdefinition und -version](#)
4. [Erstellen einer Logger-Definition und -Version](#)
5. [Abrufen des ARN der Core-Definitionsversion](#)
6. [Erstellen einer Gruppenversion](#)
7. [Eine Bereitstellung auswählen](#)
8. [Testen der Anwendung](#)

Für dieses Tutorial benötigen Sie ungefähr 30 Minuten.

Schritt 1: Erstellen Sie ein Lambda-Funktionsbereitstellungspaket

In diesem Schritt erstellen Sie ein Paket zur Bereitstellung von Lambda-Funktionen, das Python-Funktionscode und Abhängigkeiten enthält. Sie laden dieses Paket später hoch, wenn Sie die Lambda-Funktion in AWS Lambda erstellen. Die Lambda-Funktion verwendet das AWS IoT Greengrass Core SDK, um lokale Streams zu erstellen und mit ihnen zu interagieren.

Note

Ihre benutzerdefinierten Lambda-Funktionen müssen das [AWS IoT Greengrass Core SDK](#) verwenden, um mit dem Stream Manager zu interagieren. Weitere Informationen zu den Anforderungen für den Greengrass Stream-Manager finden Sie im Artikel über die [Voraussetzungen für Greengrass Stream-Manager](#).

1. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) v1.5.0 oder höher herunter.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der `greengrasssdk`-Ordner.
3. Installieren Sie Paketabhängigkeiten, die mit dem SDK in Ihr Lambda-Funktionsbereitstellungspaket aufgenommen werden sollen.
 1. Navigieren Sie zum SDK-Verzeichnis, das die `requirements.txt`-Datei enthält. Diese Datei listet die Abhängigkeiten auf.
 2. Installieren Sie die SDK-Abhängigkeiten. Führen Sie beispielsweise den folgenden `pip`-Befehl aus, um sie im aktuellen Verzeichnis zu installieren:

```
pip install --target . -r requirements.txt
```

4. Speichern Sie die folgende Pythoncode-Funktion in einer lokalen Datei namens `transfer_stream.py`.

Tip

Beispielcode, der Java und NodeJS verwendet, finden Sie im [AWS IoT Greengrass Core SDK for Java](#) und [AWS IoT Greengrass Core SDK for Node.js](#) unter. GitHub

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)

# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
# exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
# MB).
# Any data appended after the stream reaches the size limit continues to be
# appended, and
# stream manager deletes the oldest data until the total stream size is back under
# 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
# script is
# uploaded to Kinesis and you will be charged for that usage.

def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
```

```

try:
    client.delete_message_stream(stream_name=stream_name)
except ResourceNotFoundException:
    pass

exports = ExportDefinition(
    kinesisis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
)
client.create_message_stream(
    MessageStreamDefinition(
        name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
    )
)

# Append two messages and print their sequence numbers
logger.info(
    "Successfully appended message to stream with sequence number %d",
    client.append_message(stream_name, "ABCDEFGHJKLMNOP".encode("utf-8")),
)
logger.info(
    "Successfully appended message to stream with sequence number %d",
    client.append_message(stream_name, "QRSTUVWXYZ".encode("utf-8")),
)

# Try reading the two messages we just appended and print them out
logger.info(
    "Successfully read 2 messages: %s",
    client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
)

logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
# Now start putting in random data between 0 and 1000 to emulate device
sensor input
while True:
    logger.debug("Appending new random integer to stream")
    client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
    time.sleep(1)

except asyncio.TimeoutError:

```

```
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")

def function_handler(event, context):
    return

logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Packen Sie die folgenden Elemente in einer ZIP-Datei mit dem Namen `transfer_stream_python.zip`. Dies ist Ihr Lambda-Funktionsbereitstellungspaket.

- `transfer_stream.py`. App-Logik.
- `greengrasssdk`. Erforderliche Bibliothek für Python Greengrass Lambda-Funktionen, die MQTT-Nachrichten veröffentlichen.

[Stream Manager-Operationen](#) sind in Version 1.5.0 oder höher des AWS IoT Greengrass Core SDK für Python verfügbar.

- Die Abhängigkeiten, die Sie für das AWS IoT Greengrass Core SDK für Python installiert haben (z. B. die `cbor2` Verzeichnisse).

Wenn Sie die `zip`-Datei erstellen, schließen Sie nur diese Elemente ein, nicht den enthaltenen Ordner.

Schritt 2: Erstellen einer Lambda-Funktion

1. Erstellen Sie eine IAM-Rolle, damit Sie den Rollen-ARN bei der Erstellung der Funktion übergeben können.

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

        "Principal": {
            "Service": "lambda.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
]
}'

```

JSON Single-line

```

aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
"2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'

```

Note

AWS IoT Greengrass verwendet diese Rolle nicht, da die Berechtigungen für Ihre Greengrass Lambda-Funktionen in der Greengrass-Gruppenrolle angegeben sind. Für dieses Tutorial erstellen Sie eine leere Rolle.

2. Kopieren Sie die Arn aus der Ausgabe.
3. Verwenden Sie die AWS Lambda API, um die Funktion zu erstellen. TransferStream Der folgende Befehl geht davon aus, dass sich die Zip-Datei im aktuellen Verzeichnis befindet.
 - Ersetzen Sie *role-arn* durch den Arn, den Sie kopiert haben.

```

aws lambda create-function \
--function-name TransferStream \
--zip-file fileb://transfer_stream_python.zip \
--role role-arn \
--handler transfer_stream.function_handler \
--runtime python3.7

```

4. Veröffentlichen einer Version der Funktion.

```

aws lambda publish-version --function-name TransferStream --description 'First
version'

```

5. Erstellen Sie einen Alias für die veröffentlichte Version.

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

6. Kopieren Sie die `AliasArn` aus der Ausgabe. Sie verwenden diesen Wert, wenn Sie die Funktion für konfigurieren. AWS IoT Greengrass

Jetzt sind Sie bereit, die Funktion für zu konfigurieren AWS IoT Greengrass.

Schritt 3: Erstellen einer Funktionsdefinition und -version

In diesem Schritt wird eine Funktionsdefinitionsversion erstellt, die auf die `GGStreamManager` Lambda-Funktion des Systems und Ihre benutzerdefinierte `TransferStream` Lambda-Funktion verweist. Um den Stream Manager zu aktivieren, wenn Sie die AWS IoT Greengrass API verwenden, muss Ihre Funktionsdefinitionsversion die Funktion enthalten. `GGStreamManager`

1. Erstellen Sie eine Funktionsdefinition mit einer ersten Version, die die System- und benutzerdefinierten Lambda-Funktionen enthält.

Die folgende Definitionsversion aktiviert den Stream-Manager mit [Standardparametereinstellungen](#). Um benutzerdefinierte Einstellungen zu konfigurieren, müssen Sie Umgebungsvariablen für die entsprechenden Stream-Manager-Parameter definieren. Ein Beispiel finden Sie unter [the section called “Aktivieren, deaktivieren oder konfigurieren Sie den Stream Manager”](#). AWS IoT Greengrass verwendet Standardeinstellungen für Parameter, die weggelassen wurden. `MemorySize` sollte mindestens sein `128000`. `Pinned` muss auf `eingestellt` sein `true`.

Note

Eine langlebige (oder fixierte) Lambda-Funktion wird nach dem Start automatisch AWS IoT Greengrass gestartet und läuft in einem eigenen Container weiter. Dies steht im Gegensatz zu einer On-Demand-Lambda-Funktion, die startet, wenn sie aufgerufen wird, und stoppt, wenn keine Aufgaben mehr zur Ausführung übrig sind. Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

- ***arbitrary-function-id*** Ersetzen Sie es durch einen Namen für die Funktion, z. B. **stream-manager**
- ***alias-arn*** Ersetzen Sie es durch den AliasArn, den Sie kopiert haben, als Sie den Alias für die TransferStream Lambda-Funktion erstellt haben.

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
  "Functions": [
    {
      "Id": "arbitrary-function-id",
      "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
      "FunctionConfiguration": {
        "MemorySize": 128000,
        "Pinned": true,
        "Timeout": 3
      }
    },
    {
      "Id": "TransferStreamFunction",
      "FunctionArn": "alias-arn",
      "FunctionConfiguration": {
        "Executable": "transfer_stream.function_handler",
        "MemorySize": 16000,
        "Pinned": true,
        "Timeout": 5
      }
    }
  ]
}
```

```
    }
  ]
}'
```

JSON single

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-id", "FunctionArn": "arn:aws:lambda::function:GGStreamManager:1",
  "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data", "STREAM_MANAGER_SERVER_PORT":
"1234", "STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}}}, {"Id": "TransferStreamFunction",
  "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
"transfer_stream.function_handler", "MemorySize": 16000, "Pinned":
true, "Timeout": 5}}}]}'
```

Note

Timeout ist für die Funktionsdefinitionsversion erforderlich, wird vom GGStreamManager jedoch nicht verwendet. Weitere Informationen zu Timeout und anderen Einstellungen auf Gruppenebene finden Sie unter [the section called “Steuerung der Ausführung von Greengrass Lambda-Funktionen”](#)

2. Kopieren Sie die LatestVersionArn aus der Ausgabe. Mit diesem Wert fügen Sie die Version der Funktionsdefinition der Gruppenversion hinzu, die Sie im Core bereitstellen.

Schritt 4: Erstellen einer Logger-Definition und -Version

Konfigurieren Sie die Einstellungen für die Protokollierung der Gruppe. In diesem Tutorial konfigurieren Sie AWS IoT Greengrass Systemkomponenten, benutzerdefinierte Lambda-Funktionen und Konnektoren, um Protokolle in das Dateisystem des Kerngeräts zu schreiben. Sie können Protokolle verwenden, um Probleme zu beheben, die auftreten können. Weitere Informationen finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

1. Erstellen Sie eine Logger-Definition, die eine Erstversion enthält.

JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
  "Loggers": [
    {
      "Id": "1",
      "Component": "GreengrassSystem",
      "Level": "INFO",
      "Space": 10240,
      "Type": "FileSystem"
    },
    {
      "Id": "2",
      "Component": "Lambda",
      "Level": "INFO",
      "Space": 10240,
      "Type": "FileSystem"
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \
  --name "LoggingConfigs" \
  --initial-version '{"Loggers":
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSystem"},
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. Kopieren Sie den LatestVersionArn der Logger-Definition aus der Ausgabe. Mit diesem Wert fügen Sie die Version der Logger-Definition der Gruppenversion hinzu, die Sie im Core bereitstellen.

Schritt 5: Abrufen des ARN der Core-Definitionsversion

Rufen Sie den ARN der Core-Definitionsversion ab, die Sie Ihrer neuen Gruppenversion hinzufügen möchten. Um eine Gruppenversion bereitzustellen, muss sie auf eine Core-Definitionsversion verweisen, die genau einen Core enthält.

1. Rufen Sie IDs die Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

2. Kopieren Sie die Id der Zielgruppe aus der Ausgabe. Sie verwenden sie, um die Core-Definitionsversion zu erhalten, und wenn Sie die Gruppe bereitstellen.
3. Kopieren Sie die LatestVersion aus der Ausgabe, d. h. die ID der letzten Version, die der Gruppe hinzugefügt wurde. Damit erhalten Sie die Core-Definitionsversion.
4. Abrufen des ARN der Core-Definitionsversion:
 - a. Abrufen der Gruppenversion.
 - **group-id** Ersetzen Sie es durch Id das, was Sie für die Gruppe kopiert haben.
 - **group-version-id** Ersetzen Sie es durch LatestVersion das, was Sie für die Gruppe kopiert haben.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id group-version-id
```

- b. Kopieren Sie die CoreDefinitionVersionArn aus der Ausgabe. Mit diesem Wert fügen Sie die Core-Definitionsversion der Gruppenversion hinzu, die Sie im Core bereitstellen.

Schritt 6: Erstellen einer Gruppenversion

Jetzt können Sie eine Gruppenversion erstellen, die die Entitäten enthält, die Sie bereitstellen möchten. Dazu legen Sie eine Gruppenversion an, die auf die Zielversion jedes Komponententyps verweist. In diesem Lernprogramm schließen Sie eine Core-Definitionsversion, eine Funktionsdefinitionsversion und eine Logger-Definitionsversion ein.

1. Erstellen einer Gruppenversion

- *group-id* Ersetzen Sie es durch Id das, was Sie für die Gruppe kopiert haben.
- *core-definition-version-arn* Ersetzen Sie es durch CoreDefinitionVersionArn das, was Sie für die Version der Kerndefinition kopiert haben.
- *function-definition-version-arn* Ersetzen Sie es durch LatestVersionArn das, was Sie für Ihre neue Version der Funktionsdefinition kopiert haben.
- *logger-definition-version-arn* Ersetzen Sie es durch LatestVersionArn das, was Sie für Ihre neue Logger-Definitionsversion kopiert haben.

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--logger-definition-version-arn logger-definition-version-arn
```

2. Kopieren Sie die Version aus der Ausgabe. Dies ist die ID der neuen Gruppenversion.

Schritt 7: Erstellen einer Bereitstellung

Stellen Sie die Gruppe auf dem Core-Gerät bereit.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/ggc-version/bin/daemon` enthält, dann wird der Daemon ausgeführt.

 Note

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Erstellen einer -Bereitstellung.

- Ersetze es `group-id` durch Id das, was du für die Gruppe kopiert hast.
- `group-version-id` Ersetzen Sie es durch Version das, was Sie für die neue Gruppenversion kopiert haben.

```
aws greengrass create-deployment \  
--deployment-type NewDeployment \  
--group-id group-id \  
--group-version-id group-version-id
```

3. Kopieren Sie die DeploymentId aus der Ausgabe.

4. Abrufen des Bereitstellungsstatus.

- `group-id` Ersetzen Sie es durch Id das, was Sie für die Gruppe kopiert haben.
- `deployment-id` Ersetzen Sie es durch DeploymentId das, was Sie für die Bereitstellung kopiert haben.

```
aws greengrass get-deployment-status \  
--group-id group-id \  
--deployment-id deployment-id
```

Wenn der Status lautet `Success`, war die Bereitstellung erfolgreich. Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Schritt 8: Testen der Anwendung

Die `TransferStream` Lambda-Funktion generiert simulierte Gerätedaten. Sie schreibt Daten in einen Stream, den der Stream-Manager in den Kinesis-Zielfdatenstream exportiert.

1. Wählen Sie in der Amazon Kinesis Kinesis-Konsole unter Kinesis-Datenstreams die Option. `MyKinesisStream`

Note

Wenn Sie das Lernprogramm ohne Kinesis-Zielfdatenstream ausgeführt haben, [suchen Sie in der Protokolldatei](#) nach dem Stream-Manager (`GGStreamManager`). Wenn `export stream MyKinesisStream doesn't exist` in einer Fehlermeldung enthalten ist, ist der Test erfolgreich. Dieser Fehler bedeutet, dass der Service versucht hat, in den Stream zu exportieren, der Stream jedoch nicht existiert.

2. Wählen Sie auf der `MyKinesisStream`Seite `Monitoring` aus. Wenn der Test erfolgreich ist, sollten Sie die Daten in den `Put Records` (Datensätze übergeben)-Diagrammen sehen. Je nach Verbindung kann es eine Minute dauern, bis die Daten angezeigt werden.

Important

Löschen Sie nach Abschluss des Tests den Kinesis-Datenstream, um zusätzliche Kosten zu vermeiden.

Oder führen Sie die folgenden Befehle aus, um den Greengrass-Daemon zu stoppen. Dadurch wird verhindert, dass der Core Nachrichten sendet, bis Sie bereit sind, den Test fortzusetzen.

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

3. Entfernen Sie die `TransferStreamLambda`-Funktion aus dem Kern.
 - a. Folgen Sie [the section called "Erstellen einer Gruppenversion"](#), um eine neue Gruppenversion zu erstellen. Entfernen Sie jedoch die `--function-definition-version-arn`-Option im `create-group-version`-Befehl. Oder erstellen Sie eine Funktionsdefinitionsversion, die die `TransferStreamLambda`-Funktion nicht enthält.

 Note

Indem Sie die GGStreamManager Lambda-Funktion des Systems in der bereitgestellten Gruppenversion weglassen, deaktivieren Sie die Streamverwaltung auf dem Core.

- b. Folgen Sie [the section called “Eine Bereitstellung auswählen”](#), um die neue Gruppenversion bereitzustellen.

Um Protokollierungsinformationen anzuzeigen oder Probleme mit Streams zu beheben, suchen Sie in den Protokollen nach den Funktionen TransferStream und GGStreamManager. Sie müssen über root Berechtigungen zum Lesen von AWS IoT Greengrass Protokollen im Dateisystem verfügen.

- TransferStream schreibt Protokolleinträge in *greengrass-root*/ggc/var/log/user/*region*/*account-id*/TransferStream.log.
- GGStreamManager schreibt Protokolleinträge in *greengrass-root*/ggc/var/log/system/GGStreamManager.log.

Wenn Sie weitere Informationen zur Problembehandlung benötigen, können Sie die Lambda-Protokollierungsebene auf DEBUG festlegen und dann eine neue Gruppenversion erstellen und bereitstellen.

Weitere Informationen finden Sie auch unter

- [Verwalten von Daten-Streams](#)
- [the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”](#)
- [the section called “Konfigurationen für unterstützte AWS Cloud Ziele exportieren”](#)
- [the section called “Konfigurieren des Stream-Managers”](#)
- [the section called “Exportieren von Daten-Streams \(Konsole\)”](#)
- [AWS Identity and Access Management \(IAM\) -Befehle](#) in der AWS CLI Befehlsreferenz
- [AWS Lambda Befehle](#) in der AWS CLI Befehlsreferenz
- [AWS IoT Greengrass Befehle](#) in der AWS CLI Befehlsreferenz

Setzen Sie Geheimnisse bis ins AWS IoT Greengrass Mark ein

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

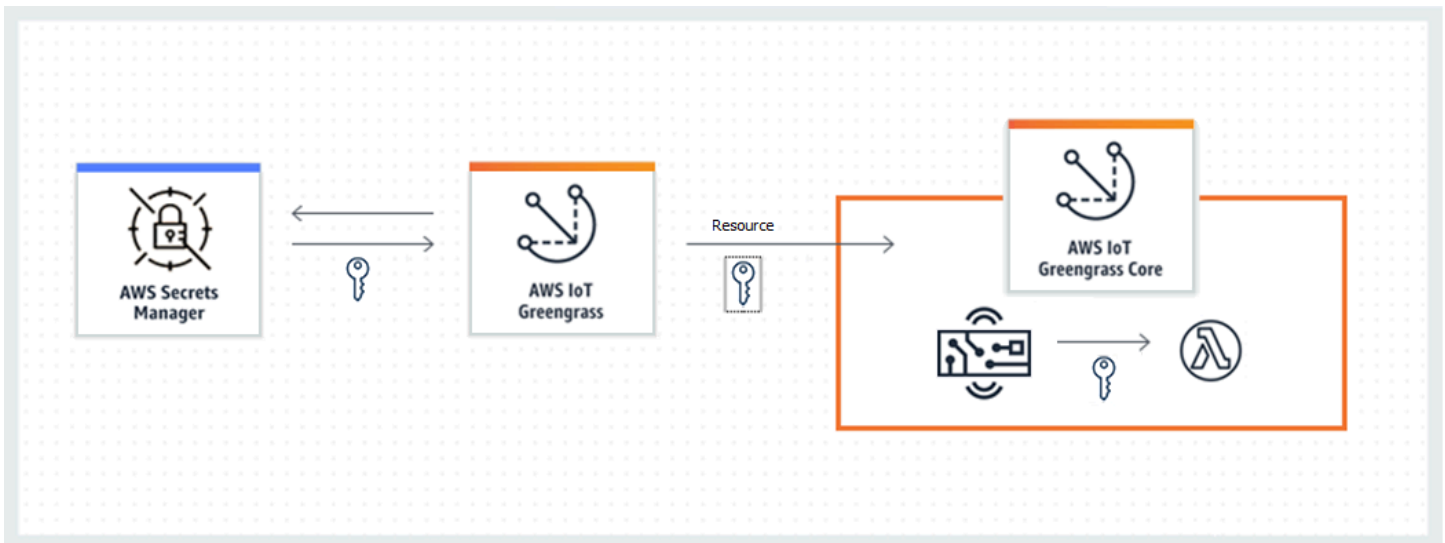
AWS IoT Greengrass ermöglicht es Ihnen, sich mit Diensten und Anwendungen von Greengrass-Geräten aus zu authentifizieren, ohne Passwörter, Token oder andere Geheimnisse fest codieren zu müssen.

AWS Secrets Manager ist ein Dienst, mit dem Sie Ihre Geheimnisse sicher in der Cloud speichern und verwalten können. AWS IoT Greengrass erweitert Secrets Manager auf Greengrass-Kerngeräte, sodass Ihre [Konnektoren](#) und Lambda-Funktionen lokale Geheimnisse verwenden können, um mit Diensten und Anwendungen zu interagieren. Beispielsweise verwendet der Twilio Notifications-Konnektor ein lokal gespeichertes Authentifizierungstoken.

Um ein Secret in eine Greengrass-Gruppe zu integrieren, erstellen Sie eine Gruppen-Ressource, die auf das Secrets Manager-Secret verweist. Diese geheime Ressource verweist über ARN auf das Cloud-Secret. Weitere Informationen zum Erstellen, Verwalten und Verwenden von Secret-Ressourcen finden Sie unter [the section called “Arbeiten mit geheimen Ressourcen”](#).

AWS IoT Greengrass verschlüsselt Ihre Geheimnisse während der Übertragung und im Ruhezustand. Ruft während der AWS IoT Greengrass Gruppenbereitstellung das Geheimnis aus Secrets Manager ab und erstellt eine lokale, verschlüsselte Kopie auf dem Greengrass-Core. Nachdem Sie Ihre Cloud-Geheimnisse in Secrets Manager rotiert haben, stellen Sie die Gruppe erneut bereit, um die aktualisierten Werte an den Core weiterzugeben.

Das folgende Diagramm zeigt den High-Level-Prozess der Bereitstellung eines Secrets im Kern. Secrets werden während der Übertragung und im Ruhezustand verschlüsselt.



Die Verwendung AWS IoT Greengrass zur lokalen Speicherung Ihrer Geheimnisse bietet folgende Vorteile:

- Entkoppelung vom Code (nicht fest programmiert). Dies unterstützt zentral verwaltete Anmeldeinformationen und schützt sensible Daten vor dem Risiko der Kompromittierung.
- Verfügbar für Offline-Szenarien. Konnektoren und Funktionen können bei fehlender Verbindung zum Internet sicher auf lokale Services und Software zugreifen.
- Kontrollierter Zugriff auf Secrets. Nur autorisierte Konnektoren und Funktionen in der Gruppe können auf Ihre Secrets zugreifen. AWS IoT Greengrass verwendet Verschlüsselung für private Schlüssel, um Ihre Secrets zu schützen. Secrets werden während der Übertragung und im Ruhezustand verschlüsselt. Weitere Informationen finden Sie unter [the section called “Verschlüsselung von Secrets”](#).
- Kontrollierte Rotation. Nachdem Sie Ihre Secrets im Secrets Manager rotiert haben, stellen Sie die Greengrass-Gruppe erneut bereit, um die lokalen Kopien Ihrer Secrets zu aktualisieren. Weitere Informationen finden Sie unter [the section called “Erstellen und Verwalten von Secrets”](#).

⚠ Important

AWS IoT Greengrass aktualisiert die Werte lokaler Geheimnisse nicht automatisch, nachdem die Cloud-Versionen rotiert wurden. Um lokale Werte zu aktualisieren, müssen Sie die Gruppe erneut bereitstellen.

Verschlüsselung von Secrets

AWS IoT Greengrass verschlüsselt Geheimnisse bei der Übertragung und im Ruhezustand.

Important

Stellen Sie sicher, dass Ihre benutzerdefinierten Lambda-Funktionen Secrets sicher behandeln und keine sensiblen Daten protokollieren, die in dem Secret gespeichert sind. Weitere Informationen finden Sie im Benutzerhandbuch unter [Minimierung der Risiken beim Protokollieren und Debuggen Ihrer Lambda-Funktion](#). AWS Secrets Manager Obwohl sich diese Dokumentation speziell auf Rotationsfunktionen bezieht, gilt die Empfehlung auch für Greengrass Lambda-Funktionen.

Verschlüsselung während der Übertragung

AWS IoT Greengrass verwendet Transport Layer Security (TLS), um die gesamte Kommunikation über das Internet und das lokale Netzwerk zu verschlüsseln. Dadurch werden Geheimnisse während der Übertragung geschützt, was passiert, wenn Geheimnisse aus Secrets Manager abgerufen und im Core bereitgestellt werden. Unterstützte TLS-Cipher-Suites finden Sie unter [the section called "Support für TLS-Verschlüsselungs-Suites"](#).

Verschlüsselung im Ruhezustand

AWS IoT Greengrass verwendet den in angegebenen privaten Schlüssel [config.json](#) für die Verschlüsselung der auf dem Core gespeicherten Geheimnisse. Aus diesem Grund ist die sichere Speicherung des privaten Schlüssels für den Schutz lokaler Secrets entscheidend. Beim [Modell der AWS geteilten Verantwortung liegt](#) es in der Verantwortung des Kunden, die sichere Speicherung des privaten Schlüssels auf dem Kerngerät zu gewährleisten.

AWS IoT Greengrass unterstützt zwei Arten der Speicherung von privaten Schlüsseln:

- Die Verwendung von Hardware-Sicherheitsmodulen. Weitere Informationen finden Sie unter [the section called "Integration von Hardware-Sicherheit"](#).

Note

AWS IoT Greengrass Unterstützt derzeit nur den [PKCS #1 v1.5-Padding-Mechanismus](#) für die Verschlüsselung und Entschlüsselung von lokalen Geheimnissen bei der Verwendung von hardwarebasierten privaten Schlüsseln. Wenn Sie die vom Hersteller

bereitgestellten Anweisungen zur manuellen Generierung hardwarebasierter privater Schlüssel befolgen, stellen Sie sicher, dass Sie PKCS #1 v1.5 wählen. AWS IoT Greengrass unterstützt Optimal Asymmetric Encryption Padding (OAEP) nicht.

- Die Verwendung von Dateisystemberechtigungen (Standard).

Der private Schlüssel wird zur Sicherung des Datenschlüssels verwendet, mit dem lokale Secrets verschlüsselt werden. Der Datenschlüssel wird bei jedem Gruppeneinsatz rotiert.

Der AWS IoT Greengrass Kern ist die einzige Entität, die Zugriff auf den privaten Schlüssel hat. Greengrass-Konnektoren oder Lambda-Funktionen, die mit einer geheimen Ressource verknüpft sind, erhalten den Wert des Geheimnisses vom Kern.

Voraussetzungen

Dies sind die Anforderungen für den lokalen Secret-Support:

- Sie müssen AWS IoT Greengrass Core v1.7 oder höher verwenden.
- Um die Werte lokaler Geheimnisse abzurufen, müssen Ihre benutzerdefinierten Lambda-Funktionen das AWS IoT Greengrass Core SDK v1.3.0 oder höher verwenden.
- Der private Schlüssel, der für die Verschlüsselung lokaler Secrets verwendet wird, muss in der Greengrass-Konfigurationsdatei angegeben sein. AWS IoT Greengrass verwendet standardmäßig den privaten Kernschlüssel, der im Dateisystem gespeichert ist. Informationen zur Bereitstellung Ihres eigenen privaten Schlüssels finden Sie unter [the section called “Angeben des privaten Schlüssels für die Verschlüsselung von Secrets”](#). Nur der RSA-Schlüsseltyp wird unterstützt.

Note

AWS IoT Greengrass unterstützt derzeit nur den [PKCS #1 v1.5-Padding-Mechanismus](#) für die Verschlüsselung und Entschlüsselung von lokalen Geheimnissen bei der Verwendung von hardwarebasierten privaten Schlüsseln. Wenn Sie die vom Hersteller bereitgestellten Anweisungen zur manuellen Generierung hardwarebasierter privater Schlüssel befolgen, stellen Sie sicher, dass Sie PKCS #1 v1.5 wählen. AWS IoT Greengrass unterstützt Optimal Asymmetric Encryption Padding (OAEP) nicht.

- AWS IoT Greengrass Ihnen muss die Erlaubnis erteilt werden, Ihre geheimen Werte abzurufen. Dies ermöglicht AWS IoT Greengrass das Abrufen der Werte während der Gruppenbereitstellung.

Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, AWS IoT Greengrass haben Sie bereits Zugriff auf Secrets mit Namen, die mit `greengrass-` beginnen. Informationen zum Anpassen des Zugriffs finden Sie unter [the section called “Erlaubt das Abrufen geheimer Werte AWS IoT Greengrass”](#).

 Note

Wir empfehlen Ihnen, diese Benennungskonvention zu verwenden, um die Geheimnisse zu identifizieren, AWS IoT Greengrass auf die zugegriffen werden darf, auch wenn Sie die Berechtigungen anpassen. Die Konsole verwendet unterschiedliche Berechtigungen, um Ihre Geheimnisse zu lesen. Es ist also möglich, dass Sie in der Konsole Geheimnisse auswählen können, für die Sie AWS IoT Greengrass keine Zugriffsberechtigung haben. Die Verwendung einer Namenskonvention kann dazu beitragen, einen Berechtigungskonflikt zu vermeiden, der zu einem Bereitstellungsfehler führt.

Angeben des privaten Schlüssels für die Verschlüsselung von Secrets

In diesem Verfahren geben Sie den Pfad zu einem privaten Schlüssel an, der für die lokale geheime Verschlüsselung verwendet wird. Dabei muss es sich um einen RSA-Schlüssel mit einer Mindestlänge von 2048 Bits handeln. Weitere Hinweise zu privaten Schlüsseln, die auf dem AWS IoT Greengrass Core verwendet werden, finden Sie unter [the section called “Sicherheitsprinzipale”](#).

AWS IoT Greengrass unterstützt zwei Modi der Speicherung privater Schlüssel: hardwarebasiert oder dateisystembasiert (Standard). Weitere Informationen finden Sie unter [the section called “Verschlüsselung von Secrets”](#).

Führen Sie diese Vorgehensweise nur dann aus, wenn Sie die Standardkonfiguration ändern möchten, die den privaten Core-Schlüssel im Dateisystem verwendet. Diese Schritte gelten unter der Annahme, dass Sie Ihre Gruppe und Ihren Kern wie in [Modul 2](#) des Tutorials Erste Schritte beschrieben erstellt haben.

1. Öffnen Sie die Datei [config.json](#), die sich im Verzeichnis `/greengrass-root/config` befindet.

 Note

greengrass-root steht für den Pfad, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

2. Geben Sie im Objekt `crypto.principals.SecretsManager` für die Eigenschaft `privateKeyPath` den Pfad des privaten Schlüssels ein:

- Wenn Ihr privater Schlüssel im Dateisystem gespeichert ist, geben Sie den absoluten Pfad zum Schlüssel an. Beispiel:

```
"SecretsManager" : {  
  "privateKeyPath" : "file:///somepath/hash.private.key"  
}
```

- Wenn Ihr privater Schlüssel in einem Hardware-Sicherheitsmodul (HSM) gespeichert ist, geben Sie den Pfad mithilfe des URI-Schemas [RFC 7512 PKCS#11](#) an. Beispiel:

```
"SecretsManager" : {  
  "privateKeyPath" : "pkcs11:object=private-key-label;type=private"  
}
```

Weitere Informationen finden Sie unter [the section called "Hardware-Sicherheitskonfiguration"](#).

 Note

AWS IoT Greengrass unterstützt derzeit nur den [PKCS #1 v1.5-Padding-Mechanismus](#) für die Verschlüsselung und Entschlüsselung von lokalen Geheimnissen bei der Verwendung von hardwarebasierten privaten Schlüsseln. Wenn Sie die vom Hersteller bereitgestellten Anweisungen zur manuellen Generierung hardwarebasierter privater Schlüssel befolgen, stellen Sie sicher, dass Sie PKCS #1 v1.5 wählen. AWS IoT Greengrass unterstützt Optimal Asymmetric Encryption Padding (OAEP) nicht.

Erlaubt das Abrufen geheimer Werte AWS IoT Greengrass

In diesem Verfahren fügen Sie der Greengrass-Servicerolle eine Inline-Richtlinie hinzu, mit der AWS IoT Greengrass Sie die Werte Ihrer Geheimnisse abrufen können.

Folgen Sie diesem Verfahren nur, wenn Sie Ihren Geheimnissen AWS IoT Greengrass benutzerdefinierte Berechtigungen gewähren möchten oder wenn Ihre Greengrass-Servicerolle die `AWSGreengrassResourceAccessRolePolicy` verwaltete Richtlinie nicht beinhaltet. `AWSGreengrassResourceAccessRolePolicy` gewährt Zugriff auf Geheimnisse, deren Namen mit `greengrass` - beginnen.

1. Führen Sie den folgenden CLI-Befehl aus, um den ARN der Greengrass-Servicerolle zu erhalten:

```
aws greengrass get-service-role-for-account --region region
```

Der zurückgegebene ARN enthält den Rollennamen.

```
{
  "AssociatedAt": "time-stamp",
  "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
}
```

Sie verwenden im folgenden Schritt den ARN oder den Namen.

2. Fügen Sie eine Inline-Richtlinie hinzu, die die Aktion `secretsmanager:GetSecretValue` erlaubt. Anweisungen finden Sie im IAM-Benutzerhandbuch unter [Hinzufügen und Entfernen von IAM-Richtlinien](#).

Sie können granularen Zugriff gewähren, indem Sie Secrets explizit auflisten oder ein Wildcard * Namensschema verwenden, oder Sie können bedingten Zugriff auf versionierte oder markierte Secrets gewähren. Mit der folgenden Richtlinie können beispielsweise nur AWS IoT Greengrass die angegebenen Geheimnisse gelesen werden.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": [
```



```
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:greengrass-SecretA-abc",
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:greengrass-SecretB-xyz"
    ]
}
]
```

Note

Wenn Sie einen vom Kunden verwalteten AWS KMS Schlüssel zum Verschlüsseln von Geheimnissen verwenden, muss Ihre Greengrass-Servicerolle die Aktion ebenfalls zulassen. `kms:Decrypt`

Weitere Informationen zu IAM-Richtlinien für Secrets Manager finden Sie im AWS Secrets Manager Benutzerhandbuch unter [Authentifizierung AWS Secrets Manager und Zugriffskontrolle für Aktionen, Ressourcen und Kontextschlüssel, die Sie in einer IAM-Richtlinie oder Geheimrichtlinie verwenden können](#). AWS Secrets Manager

Weitere Informationen finden Sie auch unter

- [Was ist? AWS Secrets Manager](#) im AWS Secrets Manager Benutzerhandbuch
- [PKCS #1: RSA Encryption Version 1.5](#)

Arbeiten mit geheimen Ressourcen

AWS IoT Greengrass verwendet geheime Ressourcen, um Geheimnisse aus AWS Secrets Manager einer Greengrass-Gruppe zu integrieren. Eine geheime Ressource ist ein Verweis auf ein Secrets Manager Manager-Geheimnis. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Auf dem AWS IoT Greengrass Kerngerät können Konnektoren und Lambda-Funktionen die geheime Ressource verwenden, um sich bei Diensten und Anwendungen zu authentifizieren, ohne dass Passwörter, Token oder andere Anmeldeinformationen fest codiert werden müssen.

Erstellen und Verwalten von Secrets

In einer Greengrass-Gruppe verweist eine geheime Ressource auf den ARN eines Secrets Manager-Geheimnisses. Wenn die geheime Ressource auf dem Core bereitgestellt wird, wird der Wert des Geheimnisses verschlüsselt und verbundenen Konnektoren und Lambda-Funktionen zur Verfügung gestellt. Weitere Informationen finden Sie unter [the section called “Verschlüsselung von Secrets”](#).

Sie verwenden Secrets Manager, um die Cloud-Versionen Ihrer Secrets zu erstellen und zu verwalten. Sie verwenden AWS IoT Greengrass es, um Ihre geheimen Ressourcen zu erstellen, zu verwalten und bereitzustellen.

Important

Wir empfehlen Ihnen, die bewährte Methode zur Rotation Ihrer Geheimnisse in Secrets Manager zu befolgen. Stellen Sie dann die Greengrass-Gruppe bereit, um die lokalen Kopien Ihrer Secrets zu aktualisieren. Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter [Rotation Ihrer AWS Secrets Manager Secrets](#).

So machen Sie ein Secret auf dem Greengrass-Core verfügbar

1. Erstellen Sie ein Geheimnis in Secrets Manager. Dies ist die Cloud-Version Ihres Geheimnisses, die zentral in Secrets Manager gespeichert und verwaltet wird. Zu den Verwaltungsaufgaben gehören das Rotieren von geheimen Werten und das Anwenden von Ressourcenrichtlinien.
2. Erstellen Sie eine geheime Ressource in AWS IoT Greengrass. Dies ist ein Gruppenressourcentyp, der über den ARN auf das Cloud-Secret verweist. Sie können ein Secret nur einmal pro Gruppe referenzieren.
3. Konfigurieren Sie Ihren Connector oder Ihre Lambda-Funktion. Sie müssen die Ressource mit einem Konnektor oder einer Funktion verknüpfen, indem Sie entsprechende Parameter oder Eigenschaften angeben. Dadurch kann von ihnen der Wert der lokal bereitgestellten geheimen Ressource ermittelt werden. Weitere Informationen finden Sie unter [the section called “Verwenden von lokalen Secrets”](#).
4. Bereitstellen der Greengrass-Gruppe Während der Bereitstellung ruft AWS IoT Greengrass den Wert des Cloud-Secrets ab und erstellt (oder aktualisiert) das lokale Secret auf dem Kern.

Secrets Manager protokolliert AWS CloudTrail jedes Mal ein Ereignis, wenn ein geheimer Wert AWS IoT Greengrass abgerufen wird. AWS IoT Greengrass protokolliert keine Ereignisse im

Zusammenhang mit der Bereitstellung oder Verwendung von lokalen Geheimnissen. Weitere Informationen zur Secrets Manager Manager-Protokollierung finden Sie unter [Überwachen der Verwendung Ihrer AWS Secrets Manager Geheimnisse](#) im AWS Secrets Manager Benutzerhandbuch.

Einbinden von Staging-Bezeichnungen in geheime Ressourcen

Secrets Manager verwendet Staging-Labels, um bestimmte Versionen eines geheimen Werts zu identifizieren. Staging-Labels können systemdefiniert oder benutzerdefiniert sein. Secrets Manager weist das AWSCURRENT Label der neuesten Version des geheimen Werts zu. Staging-Beschriftungen werden häufig verwendet, um die Rotation von Secrets zu verwalten. Weitere Informationen zur Versionierung von Secrets Manager finden Sie unter [Wichtige Begriffe und Konzepte für AWS Secrets Manager](#) im AWS Secrets Manager Benutzerhandbuch.

Geheime Ressourcen enthalten immer das AWSCURRENT Staging-Label, und sie können optional andere Staging-Labels enthalten, wenn sie von einer Lambda-Funktion oder einem Lambda-Konnektor benötigt werden. AWS IoT Greengrass Ruft während der Gruppenbereitstellung die Werte der Staging-Labels ab, auf die in der Gruppe verwiesen wird, und erstellt oder aktualisiert dann die entsprechenden Werte auf dem Core.

Erstellen und Verwalten von geheimen Ressourcen (Konsole)

Erstellen von geheimen Ressourcen (Konsole)

In der AWS IoT Greengrass Konsole erstellen und verwalten Sie geheime Ressourcen auf der Seite Ressourcen der Gruppe auf der Registerkarte Geheimnisse. Für Tutorials, die eine geheime Ressource erstellen und zu einer Gruppe hinzufügen, siehe [the section called “Erstellen einer geheimen Ressource \(Konsole\)”](#) und [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Deployments	Resources			
Subscriptions				
Cores	Local	Machine Learning	Secret	
Devices				
Lambdas				
Resources				
Connectors				
Tags				
Settings				

Add secret resource				
Resource Name ▾	Secret Name	Status	Labels	
MyTwilioAuthToken	greengrass-TwilioAuthTo...	● Unaffiliated	AWSCURRENT	...

 Note

Alternativ können Sie mit der Konsole eine geheime und geheime Ressource erstellen, wenn Sie einen Connector oder eine Lambda-Funktion konfigurieren. Sie können dies auf der Seite „Parameter konfigurieren“ des Connectors oder auf der Seite „Ressourcen“ der Lambda-Funktion tun.

Verwalten von geheimen Ressourcen (Konsole)

Zu den Verwaltungsaufgaben für die geheimen Ressourcen in Ihrer Greengrass-Gruppe gehören das Hinzufügen geheimer Ressourcen zur Gruppe, das Entfernen geheimer Ressourcen aus der Gruppe und das Ändern der [Staging-Labels](#), die in einer geheimen Ressource enthalten sind.

Wenn Sie in Secrets Manager auf ein anderes Secret verweisen, müssen Sie auch alle Connectoren bearbeiten, die das Secret verwenden:

1. Wählen Sie auf der Gruppenkonfigurationsseite die Option Connectors (Konnektoren) aus.
2. Klicken Sie im Konnektor-Kontextmenü auf Edit (Bearbeiten).
3. Die Seite Edit parameters (Parameter bearbeiten) Parameter wird eine Meldung angezeigt, um Sie zu informieren, dass sich der geheime ARN geändert hat. Wählen Sie Save (Speichern), um die Änderung zu bestätigen.

Wenn Sie ein Geheimnis in Secrets Manager löschen, entfernen Sie die entsprechende geheime Ressource aus der Gruppe und aus Konnektoren und Lambda-Funktionen, die darauf verweisen. Andernfalls wird während der Gruppenbereitstellung die Fehlermeldung AWS IoT Greengrass zurückgegeben, dass das Geheimnis nicht gefunden werden kann. Aktualisieren Sie auch Ihren Lambda-Funktionscode nach Bedarf.

Erstellen und Verwalten von geheimen Ressourcen (CLI)

Erstellen von geheimen Ressourcen (CLI)

In der AWS IoT Greengrass API ist ein Geheimnis eine Art von Gruppenressource. Das folgende Beispiel erstellt eine Ressourcendefinition mit einer Startversion, die eine geheime Ressource namens `MySecretResource` enthält. Ein Tutorial, das eine geheime Ressource erstellt und zu einer Gruppenversion hinzufügt, finden Sie unter [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#).

Die geheime Ressource verweist auf den ARN des entsprechenden Secrets Manager Manager-Geheimnisses und enthält zusätzlich zwei Staging-Labels `AWSCURRENT`, die immer enthalten sind.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
  "Resources": [
    {
      "Id": "my-resource-id",
      "Name": "MySecretResource",
      "ResourceDataContainer": {
        "SecretsManagerSecretResourceData": {
          "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
          "AdditionalStagingLabelsToDownload": [
            "Label1",
            "Label2"
          ]
        }
      }
    }
  ]
}'
```

Verwalten von geheimen Ressourcen (CLI)

Zu den Verwaltungsaufgaben für die geheimen Ressourcen in Ihrer Greengrass-Gruppe gehören das Hinzufügen geheimer Ressourcen zur Gruppe, das Entfernen geheimer Ressourcen aus der Gruppe und das Ändern der [Staging-Labels](#), die in einer geheimen Ressource enthalten sind.

In der AWS IoT Greengrass API werden diese Änderungen mithilfe von Versionen implementiert.

Die AWS IoT Greengrass API verwendet Versionen, um Gruppen zu verwalten. Versionen sind unveränderlich. Um Gruppenkomponenten — z. B. die Client-Geräte, Funktionen und Ressourcen der Gruppe — hinzuzufügen oder zu ändern, müssen Sie Versionen neuer oder aktualisierter Komponenten erstellen. Anschließend erstellen und implementieren Sie eine Gruppenversion, die die Zielversion jeder Komponente enthält. Weitere Informationen zu Gruppen finden Sie unter [the section called “AWS IoT Greengrass Gruppen”](#).

So ändern Sie beispielsweise die Staging-Beschriftungen für eine geheime Ressource:

1. Erstellen Sie eine Version einer Ressourcendefinition, die die aktualisierte geheime Ressource enthält. Das folgende Beispiel fügt der geheimen Ressource aus dem vorherigen Abschnitt eine dritte Staging-Beschriftung hinzu.

 Note

Um der Version weitere Ressourcen hinzuzufügen, nehmen Sie diese in das Array `Resources` auf.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-  
version '{  
  "Resources": [  
    {  
      "Id": "my-resource-id",  
      "Name": "MySecretResource",  
      "ResourceDataContainer": {  
        "SecretsManagerSecretResourceData": {  
          "ARN": "arn:aws:secretsmanager:us-  
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",  
          "AdditionalStagingLabelsToDownload": [  
            "Label1",  
            "Label2",  
            "Label3"  
          ]  
        }  
      }  
    }  
  ]  
}'
```

2. Wenn die ID der geheimen Ressource geändert wird, aktualisieren Sie Konnektoren und Funktionen, die die geheime Ressource verwenden. Aktualisieren Sie in den neuen Versionen den Parameter oder die Eigenschaft, die der Ressourcen-ID entspricht. Wenn der ARN des Secrets geändert wird, müssen Sie auch den entsprechenden Parameter für alle Konnektoren aktualisieren, die das Secret verwenden.

 Note

Die Ressourcen-ID ist ein beliebiger Identifikator, der vom Kunden bereitgestellt wird.

3. Erstellen Sie eine Gruppenversion, die die Zielversion jeder Komponente enthält, die Sie an den Kern senden möchten.
4. Bereitstellen der Gruppenversion.

Ein Tutorial, das zeigt, wie man geheime Ressourcen, Konnektoren und Funktionen erstellt und bereitstellt, finden Sie unter [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#).

Wenn Sie ein Geheimnis in Secrets Manager löschen, entfernen Sie die entsprechende geheime Ressource aus der Gruppe und aus Konnektoren und Lambda-Funktionen, die darauf verweisen. Andernfalls wird während der Gruppenbereitstellung die Fehlermeldung AWS IoT Greengrass zurückgegeben, dass das Geheimnis nicht gefunden werden kann. Aktualisieren Sie auch Ihren Lambda-Funktionscode nach Bedarf. Sie können ein lokales Geheimnis entfernen, indem Sie eine Version der Ressourcendefinition bereitstellen, die die entsprechende geheime Ressource nicht enthält.

Verwendung lokaler Geheimnisse in Konnektoren und Lambda-Funktionen

Greengrass-Konnektoren und Lambda-Funktionen verwenden lokale Geheimnisse, um mit Diensten und Anwendungen zu interagieren. Der AWSCURRENT-Wert wird standardmäßig verwendet, es sind aber auch Werte für andere [Staging-Bezeichnungen](#) in der Secret-Ressource verfügbar.

Konnektoren und Funktionen müssen konfiguriert werden, bevor sie auf lokale Secrets zugreifen. Dadurch wird die geheime Ressource mit dem Konnektor oder der Funktion verknüpft.

Konnektoren

Wenn ein Konnektor Zugriff auf ein lokales Secret benötigt, stellt er Parameter zur Verfügung, die Sie mit den Informationen konfigurieren, die für den Zugriff auf das Secret erforderlich sind.

- Informationen dazu, wie Sie dies in der AWS IoT Greengrass Konsole tun, finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- Informationen dazu, wie Sie dies mit der AWS IoT Greengrass CLI tun, finden Sie unter [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#).

Weitere Informationen über die Anforderungen einzelner Konnektoren finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#).

Die Logik für den Zugriff und die Nutzung des Secrets ist in den Konnektor integriert.

Lambda-Funktionen

Um einer Greengrass Lambda-Funktion den Zugriff auf ein lokales Geheimnis zu ermöglichen, konfigurieren Sie die Eigenschaften der Funktion.

- Informationen dazu, wie Sie dies in der AWS IoT Greengrass Konsole tun, finden Sie unter [the section called “Erstellen einer geheimen Ressource \(Konsole\)”](#)
- Um dies in der AWS IoT Greengrass API zu tun, geben Sie die folgenden Informationen in der `ResourceAccessPolicies` Eigenschaft an.
 - `ResourceId`: Die ID der geheimen Ressource in der Greengrass-Gruppe. Dies ist die Ressource, die auf den ARN des entsprechenden Secrets Manager Manager-Geheimnisses verweist.
 - `Permission`: Die Art des Zugriffs, den die Funktion auf die Ressource hat. Nur die Berechtigung `ro` (schreibgeschützt) wird für geheime Ressourcen unterstützt.

Das folgende Beispiel erstellt eine Lambda-Funktion, die auf die `MyApiKey` geheime Ressource zugreifen kann.

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-  
version '{  
  "Functions": [  
    {  
      "Id": "MyLambdaFunction",  
      "FunctionArn": "arn:aws:lambda:us-  
west-2:123456789012:function:myFunction:1",  
      "FunctionConfiguration": {  
        "Pinned": false,  
        "MemorySize": 16384,  
        "Timeout": 10,  
        "Environment": {  
          "ResourceAccessPolicies": [  
            {  
              "ResourceId": "MyApiKey",  
              "Permission": "ro"  
            }  
          ],  
          "AccessSysfs": true  
        }  
      }  
    }  
  ]  
}
```



```
}'
```

Um zur Laufzeit auf lokale Geheimnisse zuzugreifen, rufen Greengrass Lambda-Funktionen die `get_secret_value` Funktion vom `secretsmanager` Client im AWS IoT Greengrass Core SDK (v1.3.0 oder höher) aus.

Das folgende Beispiel zeigt, wie Sie das AWS IoT Greengrass Core SDK für Python verwenden, um ein Geheimnis abzurufen. Es übergibt den Namen des Geheimnisses an die `get_secret_value` Funktion. `SecretId` kann der Name oder der ARN des Secrets Manager-Geheimnisses sein (nicht der geheimen Ressource).

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"

def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

Für Secrets vom Typ Text gibt die `get_secret_value`-Funktion eine Zeichenfolge zurück. Für Secrets des Binärtyps gibt sie eine base64-kodierte Zeichenfolge zurück.

Important

Stellen Sie sicher, dass Ihre benutzerdefinierten Lambda-Funktionen Secrets sicher behandeln und keine sensiblen Daten protokollieren, die in dem Secret gespeichert sind. Weitere Informationen finden Sie im Benutzerhandbuch unter [Minimierung der Risiken beim Protokollieren und Debuggen Ihrer Lambda-Funktion](#). AWS Secrets Manager Obwohl sich diese Dokumentation speziell auf Rotationsfunktionen bezieht, gilt die Empfehlung auch für Greengrass Lambda-Funktionen.

Standardmäßig wird der aktuelle Wert des Secrets zurückgegeben. Dies ist die Version, der die AWSCURRENT-Staging-Bezeichnung zugeordnet ist. Um auf eine andere Version zuzugreifen, übergeben Sie den Namen der zugehörigen Staging-Bezeichnung für das optionale VersionStage-Argument. Beispiel:

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
secret_version = "MyTargetLabel"

# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

Eine weitere Beispielfunktion, die `get_secret_value` aufruft, finden Sie unter [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#).

Erstellen einer geheimen Ressource (Konsole)

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Dieses Tutorial zeigt, wie Sie eine geheime Ressource AWS-Managementkonsole zu einer Greengrass-Gruppe hinzufügen können. Eine geheime Ressource ist ein Verweis auf ein Geheimnis von AWS Secrets Manager. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Auf dem AWS IoT Greengrass Kerngerät können Konnektoren und Lambda-Funktionen die geheime Ressource verwenden, um sich bei Diensten und Anwendungen zu authentifizieren, ohne dass Passwörter, Token oder andere Anmeldeinformationen fest codiert werden müssen.

In diesem Tutorial erstellen Sie zunächst ein Geheimnis in der Konsole. AWS Secrets Manager Anschließend fügen Sie in der AWS IoT Greengrass Konsole über die Ressourcenseite der Gruppe

eine geheime Ressource zu einer Greengrass-Gruppe hinzu. Diese geheime Ressource verweist auf das Secrets Manager Manager-Geheimnis. Später fügen Sie die geheime Ressource einer Lambda-Funktion hinzu, sodass die Funktion den Wert des lokalen Geheimnisses abrufen kann.

Note

Alternativ können Sie mit der Konsole eine geheime und geheime Ressource erstellen, wenn Sie einen Connector oder eine Lambda-Funktion konfigurieren. Sie können dies auf der Seite „Parameter konfigurieren“ des Connectors oder auf der Seite „Ressourcen“ der Lambda-Funktion tun.

Nur Konnektoren, die Parameter für Secrets enthalten, können auf Secrets zugreifen. Ein Tutorial, das zeigt, wie der Twilio Notifications Connector ein lokal gespeichertes Authentifizierungstoken verwendet, finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Ein Secrets Manager Manager-Geheimnis erstellen](#)
2. [Hinzufügen einer geheimen Ressource zu einer Gruppe](#)
3. [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#)
4. [Erstellen einer Lambda-Funktion](#)
5. [Hinzufügen der -Funktion zur Gruppe](#)
6. [Anfügen der geheimen Ressource an die Funktion](#)
7. [Hinzufügen von Abonnements zur Gruppe](#)
8. [Bereitstellen der Gruppe](#)
9. [the section called “Lambda-Funktion testen”](#)

Für dieses Tutorial benötigen Sie ungefähr 20 Minuten.

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Kern (v1.7 oder höher). Weitere Informationen zum Erstellen einer Greengrass-Gruppe und Greengrass Core finden Sie unter [Erste Schritte mit AWS](#)

[IoT Greengrass](#). Das Tutorial Erste Schritte enthält auch Schritte zur Installation der AWS IoT Greengrass Core-Software.

- AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt. Weitere Informationen finden Sie unter [Anforderungen für Secrets](#).

 Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- Um die Werte lokaler Geheimnisse abzurufen, müssen Ihre benutzerdefinierten Lambda-Funktionen das AWS IoT Greengrass Core SDK v1.3.0 oder höher verwenden.

Schritt 1: Erstellen Sie ein Secrets Manager Manager-Geheimnis

In diesem Schritt verwenden Sie die AWS Secrets Manager Konsole, um ein Geheimnis zu erstellen.

1. Melden Sie sich bei der [AWS Secrets Manager -Konsole](#) an.

 Note

Weitere Informationen zu diesem Vorgang finden Sie [im AWS Secrets ManagerAWS Secrets Manager Benutzerhandbuch unter Schritt 1: Erstellen und Speichern Ihres Geheimnisses](#).

2. Wählen Sie Store a new secret (Ein neues Secret speichern).
3. Wählen Sie unter Geheimtyp auswählen die Option Anderer Geheimtyp aus.
4. Unter Specify the key/value pairs to be stored for this secret (Angabe der Schlüssel/Wert-Paare, die für dieses Secret gespeichert werden sollen):
 - Geben Sie für Key (Schlüssel) **test** ein.
 - Geben Sie für Wert **abcdefghi** ein.
5. Lassen Sie aws/secretsmanager für den Verschlüsselungsschlüssel ausgewählt und wählen Sie dann Weiter.

 Note

Es fallen keine Gebühren an, AWS KMS wenn Sie den AWS verwalteten Standardschlüssel verwenden, den Secrets Manager in Ihrem Konto erstellt.

6. Geben Sie für Secret name (Secret-Name) **greengrass-TestSecret** ein und klicken Sie auf Next (Weiter).

 Note

Standardmäßig ermöglicht AWS IoT Greengrass die Greengrass-Dienstrolle das Abrufen des Werts von Geheimnissen mit Namen, die mit greengrass - beginnen. Weitere Informationen finden Sie unter [Anforderungen für Secrets](#).

7. Für dieses Tutorial ist keine Rotation erforderlich. Wählen Sie „Automatische Rotation deaktivieren“ und anschließend „Weiter“.
8. Prüfen Sie auf der Seite Review (Prüfen) Ihre Einstellungen und wählen Sie Store (Speichern).

Als nächstes erstellen Sie in Ihrer Greengrass-Gruppe eine geheime Ressource, die auf das Secret verweist.

Schritt 2: Hinzufügen einer geheimen Ressource zu einer Greengrass-Gruppe

In diesem Schritt konfigurieren Sie eine Gruppenressource, die auf das Secrets Manager Manager-Geheimnis verweist.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe, zu der Sie die geheime Ressource hinzufügen möchten.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen und scrollen Sie dann nach unten zum Abschnitt Secrets. Im Bereich Secrets werden die geheimen Ressourcen angezeigt, die zu der Gruppe gehören. In diesem Bereich können Sie geheime Ressourcen hinzufügen, bearbeiten und entfernen.

 Note

Alternativ können Sie mit der Konsole eine geheime und geheime Ressource erstellen, wenn Sie einen Connector oder eine Lambda-Funktion konfigurieren. Sie können dies auf der Seite „Parameter konfigurieren“ des Connectors oder auf der Seite „Ressourcen“ der Lambda-Funktion tun.

4. Wählen Sie im Bereich Secrets die Option Hinzufügen aus.
5. Geben Sie auf der Seite Geheime Ressource hinzufügen **MyTestSecret** den Namen der Ressource ein.
6. Wählen Sie unter Geheim die Option greengrass- TestSecret aus.
7. Im Abschnitt Select labels (optional) steht das AWSCURRENT Staging-Label für die neueste Version des Secrets. Dieses Label ist immer in einer geheimen Ressource enthalten.

 Note

Für dieses Tutorial ist nur das AWSCURRENT Label erforderlich. Sie können optional Beschriftungen hinzufügen, die für Ihre Lambda-Funktion oder Ihren Lambda-Konnektor erforderlich sind.

8. Wählen Sie Add resource (Ressource hinzufügen) aus.

Schritt 3: Erstellen Sie ein Lambda-Funktionsbereitstellungspaket

Um eine Lambda-Funktion zu erstellen, müssen Sie zunächst ein Lambda-Funktionsbereitstellungspaket erstellen, das den Funktionscode und die Abhängigkeiten enthält. Greengrass Lambda-Funktionen benötigen das [AWS IoT Greengrass Core SDK](#) für Aufgaben wie die Kommunikation mit MQTT-Nachrichten in der Kernumgebung und den Zugriff auf lokale Geheimnisse. In diesem Tutorial wird eine Python-Funktion erstellt, sodass Sie die Python-Version des SDK im Bereitstellungspaket verwenden.

 Note

Um die Werte lokaler Geheimnisse abzurufen, müssen Ihre benutzerdefinierten Lambda-Funktionen das AWS IoT Greengrass Core SDK v1.3.0 oder höher verwenden.

1. Laden Sie von der [AWS IoT Greengrass Core SDK-Downloadseite](#) das AWS IoT Greengrass Core SDK für Python auf Ihren Computer herunter.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der greengrasssdk-Ordner.
3. Speichern Sie die folgende Pythoncode-Funktion in einer lokalen Datei namens `secret_test.py`.

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"

def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
    """
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret_value = response.get("SecretString")
    message = (
        f"Failed to retrieve secret {secret_name}."
        if secret_value is None
        else f"Successfully retrieved secret {secret_name}."
    )
    iot_client.publish(topic=send_topic, payload=message)
    print("Published: " + message)
```

Die `get_secret_value` Funktion unterstützt den Namen oder ARN des Secrets Manager Manager-Geheimnisses für den `SecretId` Wert. Dieses Beispiel verwendet den Namen des Secrets. In diesem Beispiel gibt `secret` das Schlüssel-Wert-Paar AWS IoT Greengrass zurück: `{"test": "abcdefghi"}`

 Important

Stellen Sie sicher, dass Ihre benutzerdefinierten Lambda-Funktionen Secrets sicher behandeln und keine sensiblen Daten protokollieren, die in dem Secret gespeichert sind. Weitere Informationen finden Sie im Benutzerhandbuch unter [Minimierung der Risiken beim Protokollieren und Debuggen Ihrer Lambda-Funktion](#). AWS Secrets Manager

Obwohl sich diese Dokumentation speziell auf Rotationsfunktionen bezieht, gilt die Empfehlung auch für Greengrass Lambda-Funktionen.

4. Packen Sie die folgenden Elemente in einer ZIP-Datei mit dem Namen `secret_test_python.zip`. Schließen Sie beim Erstellen einer ZIP-Datei nur den Code und die Abhängigkeiten ein, nicht den dazugehörigen Ordner.
 - `secret_test.py`. App-Logik.
 - `greengrasssdk`. Erforderliche Bibliothek für alle Python Greengrass Lambda-Funktionen.

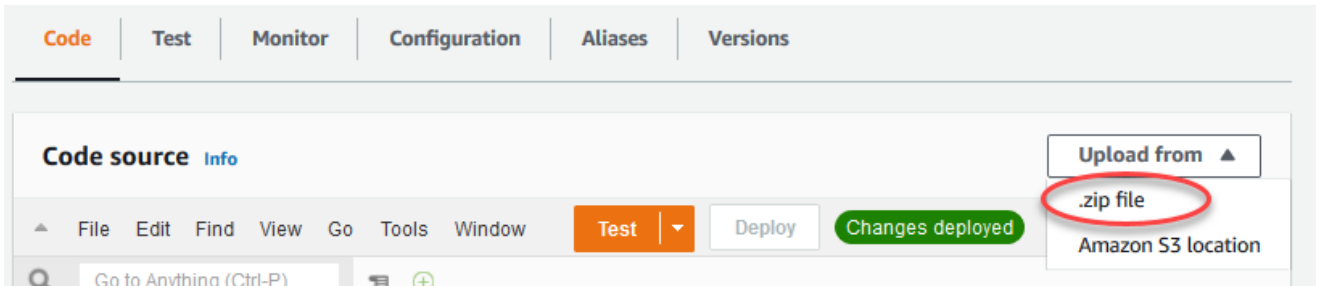
Dies ist Ihr Lambda-Funktionsbereitstellungspaket.

Schritt 4: Erstellen einer Lambda-Funktion

In diesem Schritt verwenden Sie die AWS Lambda Konsole, um eine Lambda-Funktion zu erstellen und sie für die Verwendung Ihres Bereitstellungspakets zu konfigurieren. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

1. Erstellen Sie zunächst die Lambda-Funktion.
 - a. Wählen Sie in der AWS-Managementkonsole **Dienste** aus und öffnen Sie die AWS Lambda Konsole.
 - b. Wählen Sie „Funktion erstellen“ und anschließend „Von Grund auf neu erstellen“.
 - c. Verwenden Sie im Abschnitt Basic information (Basisinformationen) folgende Werte:
 - Geben Sie für Function name (Funktionsname) **SecretTest** ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.

- d. Klicken Sie unten auf der Seite auf Create function.
2. Registrieren Sie als Nächstes den Handler und laden Sie Ihr Lambda-Funktionsbereitstellungspaket hoch.
 - a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Hochladen von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Upload und anschließend Ihr `secret_test_python.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Geben Sie unter Handler **`secret_test.function_handler`** ein.
- d. Wählen Sie Speichern.

Note

Die Testtaste auf der AWS Lambda Konsole funktioniert mit dieser Funktion nicht. Das AWS IoT Greengrass Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig in der AWS Lambda Konsole auszuführen. Diese Module (zum Beispiel `greengrass_common`) werden den Funktionen zur Verfügung gestellt, nachdem sie auf Ihrem Greengrass-Kern bereitgestellt wurden.

3. Veröffentlichen Sie jetzt die erste Version Ihrer Lambda-Funktion und erstellen Sie einen [Alias für die Version](#).

 Note

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

- a. Wählen Sie im Menü Actions die Option Publish new version aus.
- b. Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.
- c. Wählen Sie auf der Konfigurationsseite SecretTest: 1 im Menü Aktionen die Option Alias erstellen aus.
- d. Geben Sie auf der Seite Create a new alias folgende Werte an:
 - Geben Sie unter Name **GG_SecretTest** ein.
 - Wählen Sie für Version die Option 1.

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

- e. Wählen Sie Erstellen aus.

Jetzt sind Sie bereit, die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzuzufügen und die geheime Ressource anzuhängen.

Schritt 5: Fügen Sie die Lambda-Funktion zur Greengrass-Gruppe hinzu

In diesem Schritt fügen Sie die Lambda-Funktion zur Greengrass-Gruppe in der AWS IoT Konsole hinzu.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
2. Wählen Sie im Abschnitt Meine Lambda-Funktionen die Option Hinzufügen aus.

3. Wählen Sie `SecretTestSie` für die Lambda-Funktion.
4. Wählen Sie für die Lambda-Funktionsversion den Alias der Version, die Sie veröffentlicht haben.

Als Nächstes konfigurieren Sie den Lebenszyklus der Lambda-Funktion.

1. Nehmen Sie im Abschnitt Lambda-Funktionskonfiguration die folgenden Aktualisierungen vor.

 Note

Wir empfehlen, dass Sie Ihre Lambda-Funktion ohne Containerisierung ausführen, sofern Ihr Geschäftsszenario dies nicht erfordert. Auf diese Weise können Sie auf die GPU und Kamera Ihres Geräts zugreifen, ohne die Geräteressourcen konfigurieren zu müssen. Wenn Sie ohne Containerisierung arbeiten, müssen Sie auch Root-Zugriff auf Ihre AWS IoT Greengrass Lambda-Funktionen gewähren.

- a. Um ohne Containerisierung zu laufen:

- Wählen Sie für Systembenutzer und Gruppe die Option. **Another user ID/group ID**. Geben Sie als Systembenutzer-ID ein `0`. Geben Sie als Systemgruppen-ID ein `0`.

Dadurch kann Ihre Lambda-Funktion als Root ausgeführt werden. Weitere Hinweise zur Ausführung als Root finden Sie unter [the section called “Einstellung der Standardzugriffsidentität für Lambda-Funktionen in einer Gruppe”](#).

 Tip

Sie müssen Ihre `config.json` Datei auch aktualisieren, um Root-Zugriff auf Ihre Lambda-Funktion zu gewähren. Informationen zum Verfahren finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option `Kein Container aus`.

Weitere Hinweise zur Ausführung ohne Containerisierung finden Sie unter. [the section called “Überlegungen bei der Wahl der Containerisierung von Lambda-Funktionen”](#)

- Geben Sie in Timeout (Zeitüberschreitung) **10 seconds** ein.
- Wählen Sie für Pinned die Option `True` aus.

Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.

b. Um stattdessen im containerisierten Modus zu starten:

 Note

Wir empfehlen nicht, im containerisierten Modus zu arbeiten, es sei denn, Ihr Geschäftsszenario erfordert dies.

- Wählen Sie für Systembenutzer und Gruppe die Option Gruppenstandard verwenden aus.
- Wählen Sie für die Containerisierung von Lambda-Funktionen die Option Gruppenstandard verwenden aus.
- Geben Sie in Memory Limit (Speicherlimit) **1024 MB** ein.
- Geben Sie in Timeout (Zeitüberschreitung) **10 seconds** ein.
- Wählen Sie für Pinned die Option True aus.

Weitere Informationen finden Sie unter [the section called “Lebenszyklus-Konfiguration”](#).

- Wählen Sie unter Zusätzliche Parameter für Lesezugriff auf das Verzeichnis /sys die Option Aktiviert aus.

2. Wählen Sie Lambda-Funktion hinzufügen.

Ordnen Sie als Nächstes die geheime Ressource der Funktion zu.

Schritt 6: Hängen Sie die geheime Ressource an die Lambda-Funktion an

In diesem Schritt ordnen Sie die geheime Ressource der Lambda-Funktion in Ihrer Greengrass-Gruppe zu. Dadurch wird die Ressource der Funktion zugeordnet, sodass die Funktion den Wert des lokalen Geheimnisses abrufen kann.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.
2. Wählen Sie die SecretTestFunktion aus.
3. Wählen Sie auf der Detailseite der Funktion Ressourcen aus.
4. Scrollen Sie zum Bereich Secrets und wählen Sie Associate aus.

5. Wählen Sie MyTestSecret und wählen Sie dann Assoziieren aus.

Schritt 7: Hinzufügen von Abonnements zur Greengrass-Gruppe

In diesem Schritt fügen Sie Abonnements hinzu, die den Austausch von Nachrichten ermöglichen, AWS IoT und die Lambda-Funktion. Ein Abonnement ermöglicht es AWS IoT, die Funktion aufzurufen, und eines ermöglicht es der Funktion, Ausgabedaten an zu senden. AWS IoT

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.
2. Erstellen Sie ein Abonnement, mit AWS IoT dem Sie Nachrichten für die Funktion veröffentlichen können.

Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.

3. Konfigurieren Sie auf der Seite Abonnement erstellen die Quelle und das Ziel wie folgt:
 - a. Wählen Sie unter Quelltyp die Option Lambda-Funktion und dann IoT Cloud aus.
 - b. Wählen Sie unter Zieltyp die Option Service und dann aus SecretTest.
 - c. Geben Sie im Themenfilter Folgendes ein **secrets/input** und wählen Sie dann Abonnement erstellen aus.
4. Hinzufügen eines zweiten Abonnements. Wählen Sie die Registerkarte Abonnements aus, wählen Sie Abonnement hinzufügen und konfigurieren Sie Quelle und Ziel wie folgt:
 - a. Wählen Sie unter Quelltyp die Option Dienste und dann aus SecretTest.
 - b. Wählen Sie unter Zieltyp die Option Lambda-Funktion und dann IoT Cloud aus.
 - c. Geben Sie im Themenfilter Folgendes ein **secrets/output** und wählen Sie dann Abonnement erstellen aus.

Schritt 8: Bereitstellen der Greengrass-Gruppe

Stellen Sie die Gruppe auf dem Core-Gerät bereit. Ruft während der AWS IoT Greengrass Bereitstellung den Wert des Secrets aus Secrets Manager ab und erstellt eine lokale, verschlüsselte Kopie auf dem Core.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.

- a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/ggc-version/bin/daemon` enthält, dann wird der Daemon ausgeführt.

 Note

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Deploy aus.
3.
 - a. Wählen Sie auf der Registerkarte Lambda-Funktionen im Abschnitt System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
 - b. Wählen Sie im Dialogfeld IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben aus.
 - c. Wählen Sie Speichern.

Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Eine automatische Erkennung wird empfohlen, unterstützt aber AWS IoT Greengrass auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

 Note

Wenn Sie dazu aufgefordert werden, erteilen Sie die Erlaubnis, die [Greengrass-Servicerolle zu erstellen und sie Ihrer AWS-Konto aktuellen AWS-Region Rolle](#)

zuzuordnen. Diese Rolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Diensten.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss sollte der für die Bereitstellung angezeigte Status Abgeschlossen lauten.

Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Lambda-Funktion testen

1. Wählen Sie auf der Startseite der AWS IoT Konsole die Option Test aus.
2. Verwenden Sie für Thema abonnieren die folgenden Werte und wählen Sie dann Abonnieren aus.

Eigenschaft	Wert
Abonnementthema	secrets/output
MQTT-Nutzlast-Anzeige	Zeigt Nutzlasten als Zeichenfolgen an

3. Verwenden Sie für Im Thema veröffentlichen die folgenden Werte und wählen Sie dann Veröffentlichen, um die Funktion aufzurufen.

Eigenschaft	Wert
Thema	secrets/input
Fehlermeldung	Behalten der Standardnachricht. Das Veröffentlichen einer Nachricht ruft die Lambda-Funktion auf, aber die Funktion in diesem Tutorial verarbeitet den Nachricht entext nicht.

Im Erfolgsfall veröffentlicht die Funktion eine "Success"-Meldung.

Weitere Informationen finden Sie auch unter

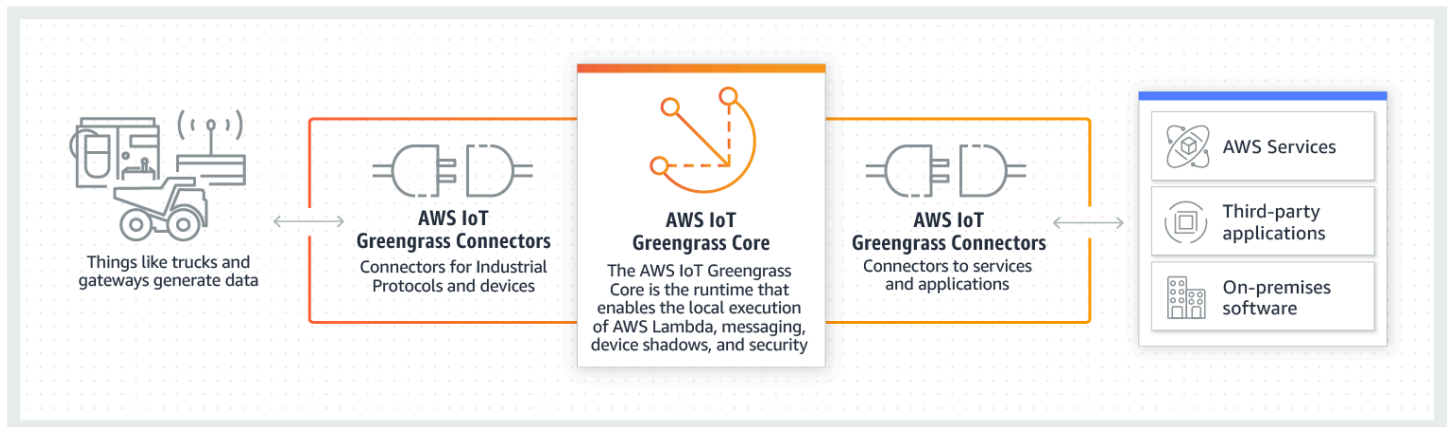
- [Bereitstellen von Secrets für den Core](#)

Integrieren von Services und Protokollen mit Greengrass-Konnektoren

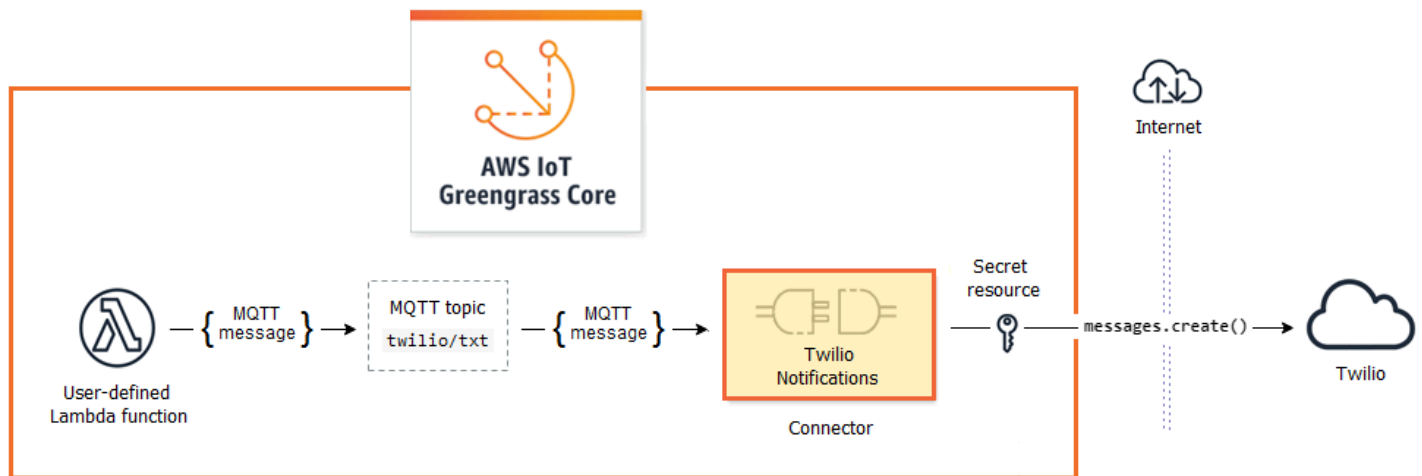
Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Bei den Konnektoren AWS IoT Greengrass handelt es sich um vorgefertigte Module, die die Interaktion mit der lokalen Infrastruktur, Geräteprotokollen und anderen Cloud-Diensten effizienter machen. AWS Durch die Verwendung von Konnektoren müssen Sie weniger Zeit mit dem Erlernen neuer Protokolle verbringen APIs und können sich mehr auf die Logik konzentrieren, die für Ihr Unternehmen wichtig ist.

Das folgende Diagramm zeigt, wo Konnektoren in die AWS IoT Greengrass Landschaft passen können.



Viele Konnektoren verwenden MQTT-Nachrichten, um mit Client-Geräten und Greengrass Lambda-Funktionen in der Gruppe oder mit dem lokalen AWS IoT Shadow-Service zu kommunizieren. Im folgenden Beispiel empfängt der Twilio Notifications Connector MQTT-Nachrichten von einer benutzerdefinierten Lambda-Funktion, verwendet eine lokale Referenz eines Geheimnisses von und ruft die AWS Secrets Manager Twilio-API auf.



Ein Tutorial zum Erstellen dieser Lösung finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#) und [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#).

Greengrass-Konnektoren können Ihnen helfen, die Gerätefunktionen zu erweitern oder Einzweckgeräte zu entwickeln. Mithilfe von Konnektoren können Sie:

- Implementieren einer wiederverwendbaren Geschäftslogik.
- Interagieren Sie mit Cloud- und lokalen Diensten, einschließlich AWS Diensten von Drittanbietern.
- Erfassen und Verarbeiten von Gerätedaten.
- Aktivieren Sie device-to-device Aufrufe mithilfe von MQTT-Themenabonnements und benutzerdefinierten Lambda-Funktionen.

AWS bietet eine Reihe von Greengrass-Konnektoren, die Interaktionen mit gängigen Diensten und Datenquellen vereinfachen. Diese vorgefertigten Module ermöglichen Szenarien für Protokollierung und Diagnose, Nachschubplanung, industrielle Datenverarbeitung sowie für Alarm und Meldung. Weitere Informationen finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#).

Voraussetzungen

Beachten Sie bei der Verwendung von Konnektoren die folgenden Punkte:

- Für jeden Konnektor, den Sie verwenden, gelten Anforderungen, die Sie erfüllen müssen. Zu diesen Anforderungen können die Mindestversion der AWS IoT Greengrass Core-Software, die

Gerätevoraussetzungen, die erforderlichen Berechtigungen und Beschränkungen gehören. Weitere Informationen finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#).

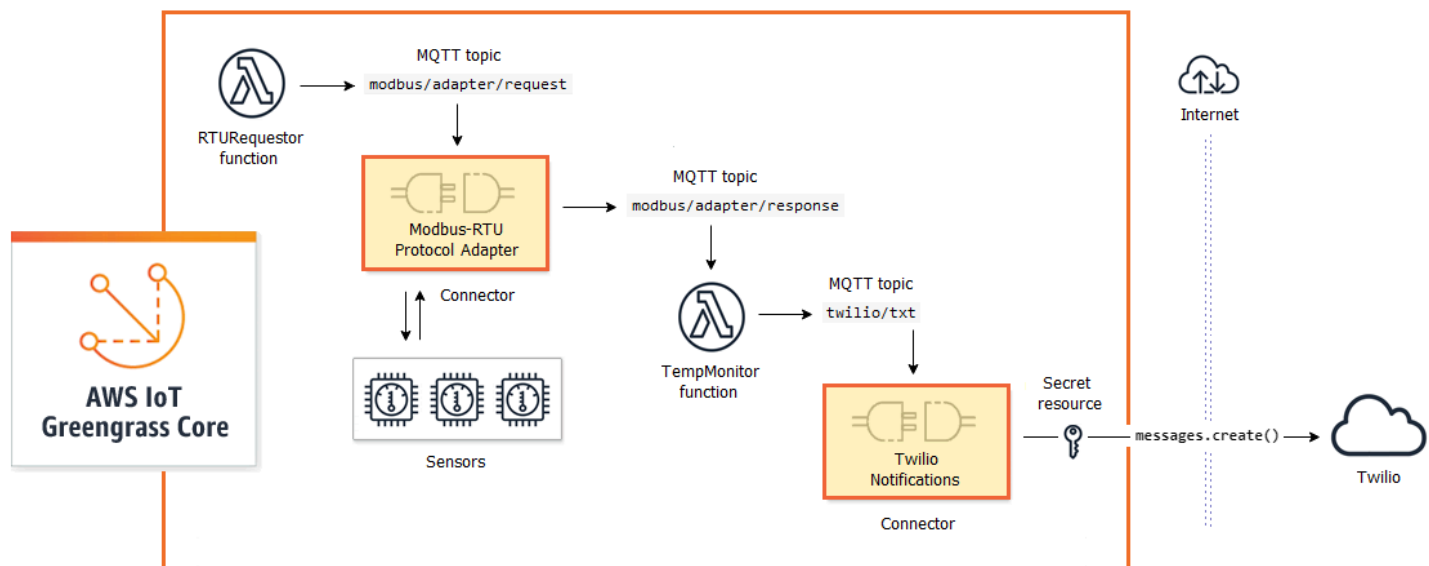
- Eine Greengrass-Gruppe kann nur eine konfigurierte Instanz eines bestimmten Connectors enthalten. Sie können die Instanz jedoch in mehreren Abonnements verwenden. Weitere Informationen finden Sie unter [the section called “Konfigurationsparameter”](#).
- Wenn die Standardcontainerisierung für die Greengrass-Gruppe auf [No container \(Kein Container\)](#) festgelegt ist, müssen die Connectors (Konnektoren) in der Gruppe ohne Containerisierung ausgeführt werden. Informationen zum Suchen von Konnektoren, die den No Container (Kein Container)-Modus unterstützen, finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#).

Verwenden von Greengrass-Konnektoren

Ein Konnektor ist ein Typ einer Gruppenkomponente. Wie bei anderen Gruppenkomponenten, wie Client-Geräten und benutzerdefinierten Lambda-Funktionen, fügen Sie Konnektoren zu Gruppen hinzu, konfigurieren ihre Einstellungen und stellen sie im AWS IoT Greengrass Core bereit. Konnektoren werden in der Core-Umgebung ausgeführt.

Sie können einige Konnektoren als einfache eigenständige Anwendungen bereitstellen. Beispielsweise liest der Device Defender-Connector Systemmetriken vom Kerngerät und sendet sie AWS IoT Device Defender zur Analyse an.

Sie können weitere Konnektoren als Bausteine zu größeren Lösungen hinzufügen. Die folgende Beispiellösung verwendet den Modbus-RTU-Protokolladapter-Konnektor zur Verarbeitung von Nachrichten von Sensoren und den Twilio Notifications Connector zur Initiierung von Twilio-Nachrichten.



Lösungen beinhalten häufig benutzerdefinierte Lambda-Funktionen, die sich neben Konnektoren befinden und die Daten verarbeiten, die der Konnektor sendet oder empfängt. In diesem Beispiel empfängt die TempMonitor Funktion Daten vom Modbus-RTU-Protokolladapter, führt einen Teil der Geschäftslogik aus und sendet dann Daten an Twilio Notifications.

Um eine Lösung zu erstellen und bereitzustellen, folgen Sie diesem allgemeinen Prozess:

1. Bilden Sie den übergeordneten Datenfluss ab. Identifizieren Sie die Datenquellen, Datenkanäle, Services, Protokolle und Ressourcen, mit denen Sie arbeiten müssen. In der Beispiellösung beinhaltet dies Daten über das Modbus RTU-Protokoll, die physikalische serielle Modbus-Schnittstelle und Twilio.
2. Identifizieren Sie die Konnektoren, die in die Lösung aufgenommen werden sollen, und fügen Sie sie zu Ihrer Gruppe hinzu. Die Beispiellösung verwendet den Modbus-RTU-Protokolladapter und Twilio-Benachrichtigungen. Damit Sie die Konnektoren leichter finden, die auf Ihr Szenario zutreffen, und um mehr über deren individuelle Anforderungen zu erfahren, konsultieren Sie die [the section called “AWS-mittelgelieferte Greengrass-Anschlüsse”](#).
3. Identifizieren Sie, ob benutzerdefinierte Lambda-Funktionen, Client-Geräte oder Ressourcen benötigt werden, und erstellen Sie sie dann und fügen Sie sie der Gruppe hinzu. Dies kann Funktionen beinhalten, die Geschäftslogik enthalten oder Daten in ein Format verarbeiten, das von einer anderen Entity in der Lösung benötigt wird. Die Beispiellösung verwendet Funktionen, um Modbus-RTU-Anfragen zu senden und Twilio-Benachrichtigungen zu initiieren. Sie enthält auch eine lokale Geräteressource für die serielle Schnittstelle Modbus RTU und eine geheime Ressource für das Twilio-Authentifizierungstoken.

 Note

Eine geheime Ressource meint Passwörter, Token und andere Secrets vom AWS Secrets Manager. Secrets können von Konnektoren und Lambda-Funktionen zur Authentifizierung bei Diensten und Anwendungen verwendet werden. AWS IoT Greengrass Kann standardmäßig auf Geheimnisse zugreifen, deren Namen mit "greengrass -" beginnen. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

4. Erstellen Sie Abonnements, die es den Entitäten in der Lösung ermöglichen, MQTT-Nachrichten auszutauschen. Wenn ein Konnektor in einem Abonnement verwendet wird, müssen der Konnektor und die Nachrichtenquelle oder das Ziel die vordefinierte Themensyntax verwenden, die von dem Konnektor unterstützt wird. Weitere Informationen finden Sie unter [the section called "Eingaben und Ausgaben"](#).
5. Bereitstellen der Gruppe auf Greengrass Core.

Informationen zum Erstellen und Bereitstellen eines Connectors finden Sie in den folgenden Tutorials:

- [the section called "Beginnen Sie mit Konnektoren \(Konsole\)"](#)
- [the section called "Erste Schritte mit Konnektoren \(CLI\)"](#)

Konfigurationsparameter

Viele Konnektoren stellen Parameter bereit, mit denen Sie Verhalten oder Ausgabe anpassen können. Diese Parameter werden während der Initialisierung, zur Laufzeit oder zu anderen Zeiten im Lebenszyklus des Konnektors verwendet.

Die Parametertypen und -verwendung variieren je nach Konnektor. Beispielsweise hat der SNS-Connector einen Parameter, der das Standard-SNS-Thema konfiguriert, und Device Defender hat einen Parameter, der die Datenabtastrate konfiguriert.

Eine Gruppenversion kann mehrere Konnektoren enthalten, aber nur eine Instance eines bestimmten Konnektor auf einmal. Dies bedeutet, dass jeder Konnektor in der Gruppe nur über eine aktive Konfiguration verfügen kann. Die Konnektor-Instance kann jedoch in mehreren Abonnements in der Gruppe verwendet werden. Sie können beispielsweise Abonnements erstellen, mit denen viele Geräte Daten an den Kinesis Firehose-Connector senden können.

Parameter für den Zugriff auf Gruppenressourcen

Greengrass-Konnektoren verwenden Gruppenressourcen für den Zugriff auf das Dateisystem, Ports, Peripheriegeräte und andere lokale Ressourcen auf dem Core-Gerät. Wenn ein Konnektor den Zugriff auf eine Gruppenressource erfordert, stellt er zugehörige Konfigurationsparameter zur Verfügung.

Gruppenressourcen beinhalten:

- [Lokale Ressourcen](#). Verzeichnisse, Dateien, Ports, Pins und Peripheriegeräte, die auf dem Greengrass Core-Gerät vorhanden sind.
- [Ressourcen für maschinelles Lernen](#). Machine Learning-Modelle, die in der Cloud geschult und für den Core für lokale Inferenzen bereitgestellt werden.
- [Geheime Ressourcen](#). Lokale, verschlüsselte Kopien von Passwörtern, Schlüsseln, Token oder beliebigem Text von. AWS Secrets Manager Konnektoren können auf diese lokalen Geheimnisse sicher zugreifen und sie verwenden, um sich auf Services oder in der lokalen Infrastruktur zu authentifizieren.

Beispielsweise ermöglichen Parameter für Device Defender den Zugriff auf Systemmetriken im /proc Host-Verzeichnis, und Parameter für Twilio-Benachrichtigungen ermöglichen den Zugriff auf ein lokal gespeichertes Twilio-Authentifizierungstoken.

Aktualisieren von Konnektorparametern

Die Parameter werden konfiguriert, wenn der Konnektor zu einer Greengrass-Gruppe hinzugefügt wird. Sie können die Parameterwerte ändern, nachdem der Konnektor hinzugefügt wurde.

- In der Konsole: Öffnen Sie auf der Gruppenkonfigurationsseite Connectors (Konnektoren) und wählen Sie im Kontextmenü des Konnektors Edit (Bearbeiten) aus.

Note

Wenn der Konnektor eine geheime Ressource verwendet, die später geändert wird, um auf ein anderes Secret zu verweisen, müssen Sie die Parameter des Konnektors bearbeiten und die Änderung bestätigen.

- In der API: Erstellen Sie eine weitere Version des Konnektors, die die neue Konfiguration definiert.

Die AWS IoT Greengrass API verwendet Versionen, um Gruppen zu verwalten. Versionen sind unveränderlich. Um Gruppenkomponenten — z. B. die Client-Geräte, Funktionen und Ressourcen der Gruppe — hinzuzufügen oder zu ändern, müssen Sie Versionen neuer oder aktualisierter Komponenten erstellen. Anschließend erstellen und implementieren Sie eine Gruppenversion, die die Zielversion jeder Komponente enthält.

Nachdem Sie Änderungen an der Konnektor-Konfiguration der Gruppe vorgenommen haben, müssen Sie die Gruppe bereitstellen, damit die Änderungen zum Core propagiert werden.

Eingaben und Ausgaben

Greengrass-Konnektoren können mit anderen Entitäten kommunizieren, indem sie MQTT-Nachrichten senden und empfangen. Die MQTT-Kommunikation wird durch Abonnements gesteuert, die es einem Connector ermöglichen, Daten mit Lambda-Funktionen, Client-Geräten und anderen Connectoren in der Greengrass-Gruppe oder mit AWS IoT und dem lokalen Shadow-Service auszutauschen. Um diese Kommunikation zuzulassen, müssen Sie Abonnements in der Gruppe erstellen, zu der der Konnektor gehört. Weitere Informationen finden Sie unter [the section called “Verwaltete Abonnements im MQTT Messaging-Workflow”](#).

Konnektoren können Nachrichtenabonnenten, Nachrichtenherausgeber oder beides sein. Jeder Konnektor definiert die MQTT-Themen, die er veröffentlicht oder abonniert. Diese vordefinierten Themen müssen in den Abonnements verwendet werden, wobei der Konnektor eine Nachrichtenquelle oder ein Nachrichtenziel ist. Tutorials mit den Schritten für die Konfiguration von Abonnements für einen Konnektor finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#) und [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#).

Note

Viele Konnektoren haben auch eingebaute Kommunikationsmodi, um mit Cloud- oder lokalen Services zu interagieren. Diese variieren je nach Konnektor und können erfordern, dass Sie Parameter konfigurieren oder der [Gruppenrolle](#) Berechtigungen hinzufügen. Weitere Informationen zu den Anforderungen für Konnektoren finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#).

Eingabethemen

Die meisten Konnektoren empfangen Eingabedaten zu MQTT-Themen. Einige Konnektoren abonnieren mehrere Themen für Eingabedaten. Der Serial Stream Connector unterstützt beispielsweise zwei Themen:

- `serial/+/read/#`
- `serial/+/write/#`

Für diesen Konnektor werden Lese- und Schreib Anforderungen an das zugehörige Thema gesendet. Wenn Sie Abonnements erstellen, stellen Sie sicher, dass Sie das Thema verwenden, das zu Ihrer Implementierung passt.

Die Zeichen `+` und `#` in den vorherigen Beispielen sind Platzhalter. Diese Platzhalter ermöglichen es Abonnenten, Nachrichten zu mehreren Themen zu empfangen, und Herausgebern, die Themen, zu denen sie veröffentlichen, anzupassen.

- Der Platzhalter `+` kann an beliebiger Stelle in der Themenhierarchie erscheinen. Es kann durch ein Hierarchie-Element ersetzt werden.

Zum Beispiel können für das Thema `sensor/+/input` Nachrichten zu den Themen `sensor/id-123/input` veröffentlicht werden, nicht aber zu `sensor/group-a/id-123/input`.

- Der `#` Platzhalter kann nur am Ende der Themenhierarchie erscheinen. Es kann durch null oder mehr Hierarchieelemente ersetzt werden..

Zum Beispiel können für das Thema `sensor/#` Nachrichten zu `sensor/`, `sensor/id-123` und `sensor/group-a/id-123` veröffentlicht werden, nicht aber zu `sensor`.

Platzhalterzeichen sind nur gültig, wenn Sie Themen abonnieren. Nachrichten können nicht zu Themen veröffentlicht werden, die Platzhalter enthalten. Weitere Informationen zu den Anforderungen an die Eingabe- oder Ausgabethemen finden Sie in der Dokumentation für den Konnektor. Weitere Informationen finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#).

Unterstützung der Containerisierung

Standardmäßig werden die meisten Konnektoren auf dem Greengrass Core in einer isolierten Laufzeitumgebung ausgeführt, die von verwaltet wird AWS IoT Greengrass. Diese Laufzeitumgebungen, Container genannt, bieten Isolation zwischen Konnektoren und dem Hostsystem. Dies erhöht die Sicherheit für den Host und den Konnektor.

Diese Greengrass-Containerisierung wird jedoch in einigen Umgebungen nicht unterstützt, z. B. wenn Sie sie AWS IoT Greengrass in einem Docker-Container oder auf älteren Linux-Kerneln ohne Cgroups ausführen. In diesen Umgebungen müssen die Konnektoren im Modus No container (Kein Container) ausgeführt werden. Informationen zum Suchen von Konnektoren, die den No Container (Kein Container)-Modus unterstützen, finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#). Einige Konnektoren werden in diesem Modus nativ ausgeführt und einige Konnektoren ermöglichen es Ihnen, den Isolationsmodus festzulegen.

Sie können den Isolationsmodus auch in Umgebungen, die die Greengrass-Containerisierung unterstützen, auf No container (Kein Container) festlegen. Wir empfehlen jedoch, wenn möglich den Modus Greengrass container (Greengrass-Container) zu verwenden.

Note

Die Standardeinstellung für Containerisierung für die Greengrass-Gruppe gilt nicht für [Konnektoren](#).

Aktualisieren von Konnektorversionen

Konnektoranbieter veröffentlichen möglicherweise neue Versionen eines Konnektors, die neue Funktionen bereitstellen, Probleme beheben oder die Leistung verbessern. Informationen zu verfügbaren Versionen und entsprechenden Änderungen finden Sie in der [Dokumentation zu den einzelnen Konnektoren](#).

In der AWS IoT Konsole können Sie nach neuen Versionen für die Konnektoren in Ihrer Greengrass-Gruppe suchen.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie unter Greengrass-Gruppen Ihre Gruppe aus.

3. Wählen Sie Connectors (Konnektoren), um die Konnektoren in der Gruppe anzuzeigen.

Wenn der Connector eine neue Version hat, wird in der Spalte Upgrade die Schaltfläche Verfügbar angezeigt.

4. So führen Sie ein Upgrade der Konnektorversion aus:

- a. Wählen Sie auf der Seite Connectors (Konnektoren) in der Spalte Upgrade die Option Available (Verfügbar). Die Seite Upgrade connector (Konnektor-Upgrade) wird geöffnet und zeigt gegebenenfalls die aktuellen Parametereinstellungen an.

Wählen Sie die neue Konnektorversion, definieren Sie die Parameter wie erforderlich und wählen Sie dann Upgrade.

- b. Fügen Sie auf der Seite Subscriptions (Abonnements) neue Abonnements in der Gruppe hinzu, um alle Abonnements zu ersetzen, die den Konnektor als Quelle oder Ziel verwenden. Entfernen Sie dann die alten Abonnements.

Abonnements verweisen nach Version auf Konnektoren, sodass sie ungültig werden, wenn Sie die Konnektorversion in der Gruppe ändern.

- c. Wählen Sie im Menü Actions (Aktionen) die Option Deploy (Bereitstellen), um Ihre Änderungen im Core bereitzustellen.

Um einen Connector über die AWS IoT Greengrass API zu aktualisieren, erstellen und implementieren Sie eine Gruppenversion, die den aktualisierten Connector und die aktualisierten Abonnements enthält. Gehen Sie genauso vor wie beim Hinzufügen eines Konnektors zu einer Gruppe. Ausführliche Schritte, die Ihnen zeigen, wie Sie den AWS CLI zum Konfigurieren und Bereitstellen eines Beispiel-Connectors für Twilio Notifications verwenden, finden Sie unter [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#).

Protokollieren für Konnektoren

Greengrass-Konnektoren enthalten Lambda-Funktionen, die Ereignisse und Fehler in Greengrass-Protokolle schreiben. Abhängig von Ihren Gruppeneinstellungen werden Protokolle in Logs, in CloudWatch das lokale Dateisystem oder in beide geschrieben. Protokolle von Konnektoren enthalten den ARN der entsprechenden Funktion. Der folgende Beispiel-ARN stammt vom Kinesis Firehose-Connector:

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

Die Standardprotokollierungskonfiguration schreibt Info-Level-Protokolle mit der folgenden Verzeichnisstruktur in das Dateisystem:

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

Weitere Hinweise zur Greengrass-Protokollierung finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

AWS-mitgelieferte Greengrass-Anschlüsse

AWS stellt die folgenden Konnektoren bereit, die allgemeine AWS IoT Greengrass Szenarien unterstützen. Weitere Informationen zur Funktionsweise von Konnektoren finden Sie in der folgenden Dokumentation:

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [Beginnen Sie mit Konnektoren \(Konsole\)](#) oder [Erste Schritte mit Konnektoren \(CLI\)](#)

Konnektor	Beschreibung	Unterstützte Lambda-Laufzeiten	Unterstützt No Container (Kein Container)-Modus
CloudWatch Metriken	Veröffentlicht benutzerdefinierte Metriken auf Amazon CloudWatch.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja
Geräteverteidiger	Sendet Systemmetriken an AWS IoT Device Defender.	• Python 3.8 * • Python 3.7 • Python 2.7	Nein
Docker-Anwendungsbereitstellung	Führt eine Docker Compose-Datei aus, um eine Docker-Anwendung auf dem Core-Gerät zu starten.	• Python 3.8 • Python 3.7	Ja

Konnektor	Beschreibung	Unterstützte Lambda-Laufzeiten	Unterstützt No Container (Kein Container)-Modus
IoT Analytics	Sendet Daten von Geräten und Sensoren an AWS IoT Analytics.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja
IoT-Ethernet-IP-Protokolladapter	Sammelt Daten von EtherNet/IP-Geräten.	• Java 8	Ja
IoT SiteWise	Sendet Daten von Geräten und Sensoren an Objekteigenschaften in AWS IoT SiteWise.	• Java 8	Ja
Kinesis Firehose	Sendet Daten an Amazon Data Firehose-Lieferdatenströme.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja
Feedback zu ML	Veröffentlicht Machine Learning-Modelleingaben in der Cloud und Ausgaben in einem MQTT-Thema.	• Python 3.8 • Python 3.7	Nein
ML-Bildklassifizierung	Führt einen lokalen Inferenzdienst zur Bildklassifizierung aus. Dieser Konnektor bietet Versionen für mehrere Plattformen.	• Python 3.8 * • Python 3.7 • Python 2.7	Nein
ML-Objekterkennung	Führt einen Inferenzdienst für die lokale Objekterkennung aus. Dieser Konnektor bietet Versionen für mehrere Plattformen.	• Python 3.8 • Python 3.7	Nein

Konnektor	Beschreibung	Unterstützte Lambda-Laufzeiten	Unterstützt No Container (Kein Container)-Modus
Modbus-RTU-Protokolladapter	Sendet Anfragen an Modbus RTU-Geräte.	• Python 3.8 * • Python 3.7 • Python 2.7	Nein
Modbus-TCP-Protokolladapter	Sammelt Daten von ModbusTCP-Geräten.	• Java 8	Ja
Raspberry Pi GPIO	Steuert GPIO-Pins auf einem Raspberry Pi-Core-Gerät.	• Python 3.8 * • Python 3.7 • Python 2.7	Nein
Serieller Stream	Liest und schreibt über eine serielle Schnittstelle eines Core-Geräts.	• Python 3.8 * • Python 3.7 • Python 2.7	Nein
ServiceNow MetricBase Integration	Veröffentlicht Zeitreihenmetriken für ServiceNow MetricBase.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja
SNS	Sendet Nachrichten an ein Amazon SNS SNS-Thema.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja

Konnektor	Beschreibung	Unterstützte Lambda-Laufzeiten	Unterstützt No Container (Kein Container)-Modus
Splunk-Integration	Veröffentlicht Daten in Splunk HEC.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja
Twilio-Benachrichtigungen	Initiiert eine Twilio-Text- oder Sprachnachricht.	• Python 3.8 * • Python 3.7 • Python 2.7	Ja

* Um die Python 3.8-Laufzeiten zu verwenden, müssen Sie einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien erstellen. Weitere Informationen finden Sie in den konnektorspezifischen Anforderungen.

Note

Wir empfehlen, [Konnektor-Versionen](#) von Python 2.7 auf Python 3.7 zu aktualisieren. Die fortgesetzte Unterstützung für Python 2.7-Konnektoren hängt von der AWS Lambda Laufzeitunterstützung ab. Weitere Informationen finden Sie unter [Richtlinien zur Runtime-Supportunterstützung](#) im AWS Lambda Entwicklerhandbuch.

CloudWatch Konnektor für Kennzahlen

Der CloudWatch [Metrics-Konnektor](#) veröffentlicht benutzerdefinierte Metriken von Greengrass-Geräten an Amazon CloudWatch. Der Connector bietet eine zentrale Infrastruktur für die Veröffentlichung von CloudWatch Metriken, mit der Sie die Greengrass-Kernumgebung überwachen und analysieren und auf lokale Ereignisse reagieren können. Weitere Informationen finden Sie unter [Verwenden von CloudWatch Amazon-Metriken](#) im CloudWatch Amazon-Benutzerhandbuch.

Dieser Konnektor empfängt Metrikdaten wie MQTT-Nachrichten. Der Connector stapelt Metriken, die sich im selben Namespace befinden, und veröffentlicht sie CloudWatch in regelmäßigen Abständen.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/CloudWatchMetrics/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 5

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

 Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `cloudwatch:PutMetricData` Aktion zulässt, wie in der folgenden Beispielrichtlinie AWS Identity and Access Management (IAM) gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Weitere Informationen zu CloudWatch Berechtigungen finden Sie in der [CloudWatch Amazon-Berechtigungsreferenz](#) im IAM-Benutzerhandbuch.

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `cloudwatch:PutMetricData` Aktion zulässt, wie in der folgenden Beispielrichtlinie AWS Identity and Access Management (IAM) gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Weitere Informationen zu CloudWatch Berechtigungen finden Sie in der [CloudWatch Amazon-Berechtigungsreferenz](#) im IAM-Benutzerhandbuch.

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Versions 4 - 5

PublishInterval

Die maximale Wartezeit in Sekunden, die bis zur Veröffentlichung von Batch-Metriken für einen bestimmten Namespace vergehen kann. Der maximale Wert beträgt 900. Um den Konnektor so zu konfigurieren, dass er Metriken veröffentlicht, sobald sie empfangen werden (ohne Batching), geben Sie 0 an.

Der Connector veröffentlicht, CloudWatch nachdem er 20 Metriken im selben Namespace oder nach dem angegebenen Intervall empfangen hat.

Note

Der Konnektor garantiert nicht die Reihenfolge der veröffentlichten Ereignisse.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall

Erforderlich: `true`

Typ: `string`

Zulässige Werte: 0 - 900

Gültiges Muster: `[0-9] | [1-9]\d | [1-9]\d\d | 900`

PublishRegion

Das AWS-Region , an das CloudWatch Metriken gepostet werden sollen. Dieser Wert überschreibt die standardmäßige Greengrass-Metrik-Region. Er ist nur bei der Veröffentlichung von regionsübergreifenden Metriken erforderlich.

Anzeigename in der AWS IoT Konsole: Region veröffentlichen

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^$ | ([a-z]{2} - [a-z]+ - \d{1})`

MemorySize

Der Speicher (in KB), der dem Konnektor zugewiesen werden soll.

Anzeigename in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

MaxMetricsToRetain

Die maximale Anzahl von Metriken über alle Namespaces hinweg, die im Speicher gespeichert werden können, bevor sie durch neue Metriken ersetzt werden. Der minimale Wert beträgt 2000.

Diese Begrenzung gilt, wenn keine Verbindung zum Internet besteht und der Konnektor beginnt, die Metriken zu puffern, um sie später zu veröffentlichen. Wenn der Puffer voll ist, werden die ältesten Metriken durch neue Metriken ersetzt. Metriken in einem bestimmten Namespace werden nur durch Metriken im selben Namespace ersetzt.

Note

Metriken werden nicht gespeichert, wenn der Host-Prozess für den Konnektor unterbrochen wird. Eine solche Unterbrechung kann beispielsweise während der Gruppenbereitstellung oder beim Neustart des Geräts auftreten.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl der beizubehaltenden Metriken

Erforderlich: true

Typ: string

Gültiges Muster: `^([2-9]\d{3}|[1-9]\d{4,})$`

IsolationMode

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: false

Typ: string

Gültige Werte: GreengrassContainer oder NoContainer.

Gültiges Muster: ^NoContainer\$|^GreengrassContainer\$

Versions 1 - 3

PublishInterval

Die maximale Wartezeit in Sekunden, die bis zur Veröffentlichung von Batch-Metriken für einen bestimmten Namespace vergehen kann. Der maximale Wert beträgt 900. Um den Konnektor so zu konfigurieren, dass er Metriken veröffentlicht, sobald sie empfangen werden (ohne Batching), geben Sie 0 an.

Der Connector veröffentlicht, CloudWatch nachdem er 20 Metriken im selben Namespace oder nach dem angegebenen Intervall empfangen hat.

Note

Der Konnektor garantiert nicht die Reihenfolge der veröffentlichten Ereignisse.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall

Erforderlich: true

Typ: string

Zulässige Werte: 0 - 900

Gültiges Muster: [0-9] | [1-9]\d | [1-9]\d\d | 900

PublishRegion

Das AWS-Region , an das CloudWatch Metriken gepostet werden sollen. Dieser Wert überschreibt die standardmäßige Greengrass-Metrik-Region. Er ist nur bei der Veröffentlichung von regionsübergreifenden Metriken erforderlich.

Anzeigename in der AWS IoT Konsole: Region veröffentlichen

Erforderlich: false

Typ: string

Gültiges Muster: `^$|([a-z]{2}-[a-z]+-\d{1})`

MemorySize

Der Speicher (in KB), der dem Konnektor zugewiesen werden soll.

Anzeigename in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

MaxMetricsToRetain

Die maximale Anzahl von Metriken über alle Namespaces hinweg, die im Speicher gespeichert werden können, bevor sie durch neue Metriken ersetzt werden. Der minimale Wert beträgt 2000.

Diese Begrenzung gilt, wenn keine Verbindung zum Internet besteht und der Konnektor beginnt, die Metriken zu puffern, um sie später zu veröffentlichen. Wenn der Puffer voll ist, werden die ältesten Metriken durch neue Metriken ersetzt. Metriken in einem bestimmten Namespace werden nur durch Metriken im selben Namespace ersetzt.

Note

Metriken werden nicht gespeichert, wenn der Host-Prozess für den Konnektor unterbrochen wird. Eine solche Unterbrechung kann beispielsweise während der Gruppenbereitstellung oder beim Neustart des Geräts auftreten.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl der beizubehaltenden Metriken

Erforderlich: true

Typ: string

Gültiges Muster: `^([2-9]\d{3}|[1-9]\d{4,})$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den CloudWatch Metrics-Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyCloudWatchMetricsConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/  
versions/4",  
      "Parameters": {  
        "PublishInterval" : "600",  
        "PublishRegion" : "us-west-2",  
        "MemorySize" : "16",  
        "MaxMetricsToRetain" : "2500",  
        "IsolationMode" : "GreengrassContainer"  
      }  
    }  
  ]  
}'
```

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert Metriken zu einem MQTT-Thema und veröffentlicht die Metriken unter. CloudWatch Eingabenachrichten müssen im JSON-Format sein.

Themenfilter im Abonnement

```
cloudwatch/metric/put
```

Nachrichten-Eigenschaften

```
request
```

Informationen über die Metrik in dieser Meldung.

Das Anforderungsobjekt enthält die metrischen Daten, die an CloudWatch veröffentlicht werden sollen. Die Metrikwerte müssen den Spezifikationen der [PutMetricDataAPI](#)

entsprechen. Es werden nur die Eigenschaften `namespace`, `metricData.metricName` und `metricData.value` benötigt.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`namespace`

Der benutzerdefinierte Namespace für die Metrikdaten in dieser Anfrage. CloudWatch verwendet Namespaces als Container für metrische Datenpunkte.

 Note

Sie können keinen Namespace angeben, der mit der reservierten Zeichenfolge beginnt. `AWS/`

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^[^:]*`

`metricData`

Die Daten für die Metrik.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`metricName`

Name der Metrik.

Erforderlich: `true`

Typ: `string`

`dimensions`

Die Dimensionen, die der Metrik zugeordnet sind. Dimensionen liefern zusätzliche Informationen über die Metrik und ihre Daten. Eine Metrik kann bis zu 10 Dimensionen definieren.

Dieser Konnektor enthält automatisch eine Dimension mit dem NamenscoreName, wobei der Wert dem Namen des Kerns entspricht.

Erforderlich: false

Typ: array von Dimensionsobjekten, die die folgenden Eigenschaften enthalten:

name

Der Dimensionsname.

Erforderlich: false

Typ: string

value

Der Dimensionswert.

Erforderlich: false

Typ: string

timestamp

Die Zeit, zu der die metrischen Daten empfangen wurden, ausgedrückt als Anzahl der Sekunden seit dem Jan 1, 1970 00:00:00 UTC. Wenn dieser Wert weggelassen wird, verwendet der Konnektor den Zeitpunkt an dem er die Nachricht empfangen hat.

Erforderlich: false

Typ: timestamp

 Note

Wenn Sie zwischen den Versionen 1 und 4 dieses Connectors verwenden, empfehlen wir, den Zeitstempel für jede Metrik separat abzurufen, wenn Sie mehrere Metriken aus einer einzigen Quelle senden. Verwenden Sie keine Variable, um den Zeitstempel zu speichern.

value

Der Wert für die Metrik.

 Note

CloudWatch lehnt zu kleine oder zu große Werte ab. Die Werte müssen im Bereich von $8.515920e-109$ bis $1.174271e+108$ (Base 10) oder $2e-360$ bis $2e360$ (Base 2) liegen. Spezielle Werte (z. B. NaN, +Infinity, -Infinity) werden nicht unterstützt.

Erforderlich: true

Typ: double

unit

Die Einheit der Metrik.

Erforderlich: false

Typ: string

Zulässige Werte: Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/Second, Terabits/Second, Count/Second, None

Einschränkungen

Alle von der CloudWatch [PutMetricData](#) API auferlegten Beschränkungen gelten für Metriken, wenn dieser Connector verwendet wird. Von besonderer Bedeutung sind die folgenden Grenzwerte:

- 40 KB Limit auf API-Nutzlast
- 20 Metriken pro API-Anforderung
- 150 Transaktionen pro Sekunde (TPS) für die PutMetricData-API

Weitere Informationen finden Sie unter [CloudWatch Beschränkungen](#) im CloudWatch Amazon-Benutzerhandbuch.

Beispieleingabe

```
{
```

```
"request": {
  "namespace": "Greengrass",
  "metricData":
    {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
}
```

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema.

Themenfilter im Abonnement

cloudwatch/metric/put/status

Beispielausgabe: Erfolg

Die Antwort umfasst den Namespace der Metrikdaten und das RequestId Feld aus der CloudWatch Antwort.

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

Beispielausgabe: Fehler

```
{
```

```
"response" : {  
    "namespace": "Greengrass",  
    "error": "InvalidInputException",  
    "error_message": "cw metric is invalid",  
    "status": "fail"  
}  
}
```

Note

Wenn der Connector einen Fehler erkennt, der wiederholt werden kann (z. B. Verbindungsfehler), versucht er erneut, die Veröffentlichung im nächsten Batch durchzuführen.

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": true in der CLI).
 - b. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - c. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": false in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
import time
import json
```

```
iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
    return {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
                       payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der CloudWatch Metrics-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain

- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
5	Der Fehler wurde behoben, um Unterstützung für doppelte Zeitstempel in Eingabedaten hinzuzufügen.
4	Der Parameter <code>IsolationMode</code> wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)

- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Verwendung von CloudWatch Amazon-Metriken](#) im CloudWatch Amazon-Benutzerhandbuch
- [PutMetricData](#) in der Amazon CloudWatch API-Referenz

Device Defender-Anschluss

Der Device [Defender-Connector](#) benachrichtigt Administratoren über Statusänderungen eines Greengrass-Core-Geräts. Dies kann helfen, ungewöhnliches Verhalten zu erkennen, das auf ein gefährdetes Gerät hinweisen könnte.

Dieser Connector liest Systemmetriken aus dem `/proc` Verzeichnis auf dem Kerngerät und veröffentlicht die Metriken dann auf AWS IoT Device Defender Einzelheiten zur Berichterstattung über Metriken finden Sie in der [Dokumentspezifikation zu Gerätemetriken](#) im AWS IoT Entwicklerhandbuch.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/DeviceDefender/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/DeviceDefender/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/DeviceDefender/versions/1</code>

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- AWS IoT Device Defender konfiguriert, um die Erkennungsfunktion zu verwenden, um Verstöße zu verfolgen. Weitere Informationen finden Sie unter [Detect](#) im AWS IoT Entwicklerhandbuch.
- Eine [lokale Volume-Ressource](#) in der Greengrass-Gruppe, die auf das /proc Verzeichnis verweist. Die Ressource muss die folgenden Eigenschaften verwenden:
 - Quellpfad: /proc
 - Zielpfad: /host_proc (oder einen Wert, der dem [gültigen Muster](#) entspricht)
 - AutoAddGroupOwner: true
- Die [psutil](#)-Bibliothek muss auf dem Greengrass Core installiert sein. Version 5.7.0 ist die neueste Version, die nachgewiesenermaßen mit dem Konnektor funktioniert.
- Die [cbor](#)-Bibliothek muss auf dem Greengrass Core installiert sein. Version 1.0.0 ist die neueste Version, die nachgewiesenermaßen mit dem Konnektor funktioniert.

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- AWS IoT Device Defender konfiguriert, um die Erkennungsfunktion zu verwenden, um Verstöße zu verfolgen. Weitere Informationen finden Sie unter [Detect](#) im AWS IoT Entwicklerhandbuch.

- Eine [lokale Volume-Ressource](#) in der Greengrass-Gruppe, die auf das `/proc` Verzeichnis verweist. Die Ressource muss die folgenden Eigenschaften verwenden:
 - Quellpfad: `/proc`
 - Zielpfad: `/host_proc` (oder einen Wert, der dem [gültigen Muster](#) entspricht)
 - `AutoAddGroupOwner`: `true`
- Die [psutil](#)-Bibliothek muss auf dem Greengrass Core installiert sein.
- Die [cbor](#)-Bibliothek muss auf dem Greengrass Core installiert sein.

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

SampleIntervalSeconds

Die Anzahl der Sekunden zwischen jedem Zyklus der Erfassung und Berichterstattung von Metriken. Die Mindestwert beträgt 300 Sekunden (5 Minuten).

Anzeigenname in der AWS IoT Konsole: Berichtsintervall für Metriken

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

ProcDestinationPath-ResourceId

Die ID der Volume-Ressource `/proc`.

Note

Dieser Konnektor hat nur Lesezugriff auf die Ressource.

Anzeigenname in der AWS IoT Konsole: Ressource für das Verzeichnis `/proc`

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `[a-zA-Z0-9_-]+`

ProcDestinationPath

Der Ziel-Pfad der Volume-Ressource /proc.

Anzeigenname in der AWS IoT Konsole: Zielpfad der /proc-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: `\/[a-zA-Z0-9_-]+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den Device Defender-Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyDeviceDefenderConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/  
versions/3",  
      "Parameters": {  
        "SampleIntervalSeconds": "600",  
        "ProcDestinationPath": "/host_proc",  
        "ProcDestinationPath-ResourceId": "my-proc-resource"  
      }  
    }  
  ]  
}'
```

Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Connector veröffentlicht Sicherheitsmetriken AWS IoT Device Defender als Ausgabedaten.

Themenfilter im Abonnement

```
$aws/things/+/defender/metrics/json
```

Note

Dies ist die Themensyntax, die AWS IoT Device Defender erwartet wird. Der Konnektor ersetzt den +-Platzhalter durch den Gerätenamen (z. B. `$aws/things/thing-name/defender/metrics/json`).

Beispielausgabe

Einzelheiten zur Berichterstattung über Metriken finden Sie in der [Dokumentenspezifikation für Gerätemetriken](#) im AWS IoT Entwicklerhandbuch.

```
{
  "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
    }
}
}
}
}

```

Lizenzen

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Device Defender](#) im AWS IoT Entwicklerhandbuch

Docker-Anwendungsbereitstellungs-Konnektor

Der Greengrass Docker Application Deployment Connector macht es einfacher, Ihre Docker-Images auf einem Core auszuführen. AWS IoT Greengrass Der Konnektor verwendet Docker Compose, um eine Docker-Anwendung mit mehreren Containern aus einer `docker-compose.yml`-Datei zu starten. Insbesondere führt der Konnektor `docker-compose`-Befehle aus, um Docker-Container auf

einem Single-Core-Gerät zu verwalten. Weitere Informationen finden Sie unter [Übersicht über Docker Compose](#) in der Docker-Dokumentation. Der Connector kann auf Docker-Images zugreifen, die in Docker-Container-Registern wie Amazon Elastic Container Registry (Amazon ECR), Docker Hub und privaten vertrauenswürdigen Docker-Registern gespeichert sind.

Nachdem Sie die Greengrass-Gruppe bereitgestellt haben, ruft der Connector die neuesten Images ab und startet die Docker-Container. Er führt den Befehl `and ausdocker-compose pull. docker-compose up` Anschließend veröffentlicht der Konnektor den Status des Befehls in einem [MQTT-Ausgabethema](#). Er protokolliert auch Statusinformationen zum Ausführen von Docker-Containern. Auf diese Weise können Sie Ihre Anwendungsprotokolle in Amazon überwachen CloudWatch. Weitere Informationen finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#). Der Konnektor startet auch Docker-Container bei jedem Neustart des Greengrass-Daemons. Die Anzahl der Docker-Container, die auf dem Core ausgeführt werden können, hängt von Ihrer Hardware ab.

Die Docker-Container werden außerhalb der Greengrass-Domäne auf dem Core-Gerät ausgeführt, sodass sie nicht auf die Interprozesskommunikation (IPC) des Cores zugreifen können. Sie können jedoch einige Kommunikationskanäle mit Greengrass-Komponenten konfigurieren, z. B. lokale Lambda-Funktionen. Weitere Informationen finden Sie unter [the section called “Kommunikation mit Docker-Containern”](#).

Sie können den Konnektor für Szenarien wie das Hosten eines Webserver oder eines MySQL-Servers auf Ihrem Core-Gerät verwenden. Lokale Services in Ihren Docker-Anwendungen können miteinander, mit anderen Prozessen in der lokalen Umgebung und mit Cloud-Services kommunizieren. Sie können beispielsweise einen Webserver auf dem Core ausführen, der Anfragen von Lambda-Funktionen an einen Webdienst in der Cloud sendet.

Dieser Konnektor wird im [No container \(Kein Container\)](#)-Isolationsmodus ausgeführt, so dass Sie ihn für eine Greengrass-Gruppe bereitstellen können, die ohne Greengrass-Containerisierung ausgeführt wird.

Dieser Connector hat die folgenden Versionen.

Version	ARN
7	<code>arn:aws:greengrass: <i>region</i>::/connectors/DockerApplicationDeployment/versions/7</code>

Version	ARN
6	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/6
5	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/DockerApplicationDeployment/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

- AWS IoT Greengrass Kernsoftware v1.10 oder höher.

Note

Dieser Connector wird auf OpenWrt Distributionen nicht unterstützt.

- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

 Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Mindestens 36 MB RAM auf dem Greengrass Core für den Konnektor, um laufende Docker-Container zu überwachen. Der Gesamtspeicherbedarf hängt von der Anzahl der Docker-Container ab, die auf dem Core ausgeführt werden.
- [Docker Engine](#) 1.9.1 oder höher muss auf dem Greengrass Core installiert sein. Version 19.0.3 ist die neueste Version, die nachgewiesenermaßen mit dem Konnektor funktioniert.

Die ausführbare docker-Datei muss sich im Verzeichnis /usr/bin oder /usr/local/bin befinden.

 Important

Es wird empfohlen, einen Anmeldeinformationsspeicher zu installieren, um die lokalen Kopien Ihrer Docker-Anmeldeinformationen zu sichern. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Informationen zur Installation von Docker auf Amazon Linux-Distributionen finden Sie unter [Docker-Grundlagen für Amazon ECS im Amazon](#) Elastic Container Service Developer Guide.

- [Docker Compose](#) installiert auf dem Greengrass Core. Die ausführbare docker-compose-Datei muss sich im Verzeichnis /usr/bin oder /usr/local/bin befinden.

Die folgenden Docker Compose-Versionen funktionieren nachgewiesenermaßen mit dem Konnektor.

Konnektor-Version	Verifizierte Docker Compose-Version
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- Eine einzelne Docker Compose-Datei (z. B. `docker-compose.yml`), gespeichert in Amazon Simple Storage Service (Amazon S3). Das Format muss mit der auf dem Core installierten Version von Docker Compose kompatibel sein. Sie sollten die Datei testen, bevor Sie sie auf Ihrem Core verwenden. Wenn Sie die Datei bearbeiten, nachdem Sie die Greengrass-Gruppe bereitgestellt haben, müssen Sie die Gruppe erneut bereitstellen, um Ihre lokale Kopie auf dem Core zu aktualisieren.
- Ein Linux-Benutzer mit der Berechtigung, den lokalen Docker-Daemon aufzurufen und in das Verzeichnis zu schreiben, in dem die lokale Kopie Ihrer Compose-Datei gespeichert ist. Weitere Informationen finden Sie unter [Einrichten des Docker-Benutzers auf dem Core](#).
- Die [Greengrass-Gruppenrolle](#) muss so konfiguriert sein, dass sie die Aktion `s3:GetObject` für den S3-Bucket zulässt, der Ihre Compose-Datei enthält. Diese Berechtigung wird in der folgenden Beispiel-IAM-Richtlinie gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessToComposeFileS3Bucket",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ]
    }
  ]
}
```

```
    ],  
    "Effect": "Allow",  
    "Resource": "arn:aws:s3:::bucket-name/*"  
  }  
]  
}
```

Note

Wenn Ihr S3-Bucket versioning-fähig ist, muss die Rolle the so konfiguriert werden, dass sie die `s3:GetObjectVersion` Aktion ebenfalls zulässt. Weitere Informationen finden Sie unter [Verwenden der Versionierung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

- Wenn Ihre Docker Compose-Datei auf ein in Amazon ECR gespeichertes Docker-Image verweist, ist die [Greengrass-Gruppenrolle](#) so konfiguriert, dass sie Folgendes zulässt:
 - `ecr:GetDownloadUrlForLayer` und `ecr:BatchGetImage` Aktionen in Ihren Amazon ECR-Repositorys, die die Docker-Images enthalten.
 - Die `ecr:GetAuthorizationToken`-Aktion für die Ressourcen.

Repositorys müssen sich im selben AWS-Konto und AWS-Region wie der Connector befinden.

Important

Berechtigungen in der Gruppenrolle können von allen Lambda-Funktionen und -Konnektoren in der Greengrass-Gruppe übernommen werden. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Diese Berechtigungen werden in der folgenden Beispielrichtlinie angezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowGetEcrRepositories",
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": [
        "arn:aws:ecr:us-east-1:123456789012:repository/repository-name"
      ]
    },
    {
      "Sid": "AllowGetEcrAuthToken",
      "Effect": "Allow",
      "Action": "ecr:GetAuthorizationToken",
      "Resource": "*"
    }
  ]
}
```

Weitere Informationen finden Sie unter [Beispiele für Amazon ECR-Repository-Richtlinien](#) im Amazon ECR-Benutzerhandbuch.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

- Wenn Ihre Docker Compose-Datei auf ein Docker-Abbild aus [AWS Marketplace](#) verweist, gelten für den Konnektor außerdem folgende Anforderungen:
 - Sie müssen Container-Produkte abonniert haben. AWS Marketplace Weitere Informationen finden Sie unter [Suchen und Abonnieren von Containerprodukten](#) im AWS Marketplace - Abonnentenhandbuch.

- AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben. Der Connector verwendet diese Funktion nur, um Ihre Geheimnisse abzurufen AWS Secrets Manager, nicht um sie zu speichern.
- Sie müssen in Secrets Manager für jede AWS Marketplace Registrierung, die ein Docker-Image speichert, auf das in Ihrer Compose-Datei verwiesen wird, ein Geheimnis erstellen. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositorys”](#).
- Wenn Ihre Docker Compose-Datei auf ein Docker-Image aus privaten Repositorys in anderen Registern als Amazon ECR verweist, z. B. Docker Hub, hat der Connector auch die folgenden Anforderungen:
 - AWS IoT Greengrass [muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in Secrets Requirements beschrieben](#). Der Connector verwendet diese Funktion nur, um Ihre Geheimnisse abzurufen AWS Secrets Manager, nicht um sie zu speichern.
 - Sie müssen in Secrets Manager für jedes private Repository, das ein Docker-Image speichert, auf das in Ihrer Compose-Datei verwiesen wird, ein Geheimnis erstellen. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositorys”](#).
- Der Docker-Daemon muss ausgeführt werden, wenn Sie eine Greengrass-Gruppe bereitstellen, die diesen Konnektor enthält.

Zugreifen auf Docker-Images aus privaten Repositorys

Wenn Sie Anmeldeinformationen verwenden, um auf Ihre Docker-Abbilder zuzugreifen, müssen Sie dem Konnektor erlauben, darauf zuzugreifen. Wie Sie dies tun, hängt davon ab, wo sich das Docker-Abbild befindet.

Für Docker-Images, die in Amazon ECR gespeichert sind, erteilen Sie die Erlaubnis, Ihr Autorisierungstoken in der Greengrass-Gruppenrolle abzurufen. Weitere Informationen finden Sie unter [the section called “Voraussetzungen”](#).

Für Docker-Images, die in anderen privaten Repositorys oder Registern gespeichert sind, müssen Sie ein Secret erstellen, in AWS Secrets Manager dem Ihre Anmeldeinformationen gespeichert werden. Dazu gehören Docker-Images, die Sie abonniert haben. AWS Marketplace Erstellen Sie ein Secret für jedes Repository. Wenn Sie Ihre Secrets in Secrets Manager aktualisieren, werden die Änderungen bei der nächsten Bereitstellung der Gruppe auf den Core übertragen.

Note

Secrets Manager ist ein Dienst, mit dem Sie Ihre Anmeldeinformationen, Schlüssel und andere Geheimnisse sicher speichern und verwalten können AWS Cloud. Weitere Informationen finden Sie unter [Was ist AWS Secrets Manager?](#) im AWS Secrets Manager Benutzerhandbuch.

Jedes Secret muss die folgenden Schlüssel enthalten:

Key (Schlüssel)	Value (Wert)
username	Der Benutzername, der für den Zugriff auf das Repository oder die Registrierung verwendet wird.
password	Das Passwort, das für den Zugriff auf das Repository oder die Registrierung verwendet wird.
registryUrl	Der Endpunkt der Registrierung. Dies muss mit der entsprechenden Registrierungs-URL in der Compose-Datei übereinstimmen.

Note

Um standardmäßig auf ein Secret zugreifen AWS IoT Greengrass zu können, muss der Name des Secrets mit greengrass- beginnen. Andernfalls muss Ihre Greengrass-Servicerolle Zugriff gewähren. Weitere Informationen finden Sie unter [the section called “Erlaubt das Abrufen geheimer Werte AWS IoT Greengrass”](#).

So rufen Sie Anmeldeinformationen für Docker-Abbilder aus AWS Marketplace ab

1. Rufen Sie Ihr Passwort für Docker-Images mit dem AWS Marketplace `aws ecr get-login-password` Befehl ab. Weitere Informationen finden Sie unter [get-login-password](#) in der Referenz zum AWS CLI -Befehl.

```
aws ecr get-login-password
```

2. Rufen Sie die Registrierungs-URL für das Docker-Image ab. Öffnen Sie die AWS Marketplace Website und die Startseite für das Container-Produkt. Wählen Sie unter Container-Images die Option Container-Image-Details anzeigen aus, um den Benutzernamen und die Registrierungs-URL zu finden.

Verwenden Sie den abgerufenen Benutzernamen, das Passwort und die Registrierungs-URL, um ein Geheimnis für jede AWS Marketplace Registrierung zu erstellen, in der Docker-Images gespeichert werden, auf die in Ihrer Compose-Datei verwiesen wird.

So erstellen Sie Secrets (Konsole)

Wählen Sie in der AWS Secrets Manager Konsole Andere Arten von Geheimnissen aus. Fügen Sie unter Angabe der Schlüssel/Wert-Paare, die für dieses Secret gespeichert werden sollen Zeilen für `username`, `password` und `registryUrl` hinzu. Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter [Erstellen eines grundlegenden Geheimnisses](#).

Specify the key/value pairs to be stored in this secret [Info](#)

Secret key/value	Plaintext	
username	Mary_Major	Remove
password	abc123xyz456	Remove
registryUrl	https://docker.io	Remove

[+ Add row](#)

So erstellen Sie Secrets (CLI)

Verwenden Sie in der AWS CLI den Secrets Manager `create-secret` Manager-Befehl, wie im folgenden Beispiel gezeigt. Weitere Informationen finden Sie unter [create-secret](#) in der AWS CLI Befehlsreferenz.

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string  
[{"username":"Mary_Major"}, {"password":"abc123xyz456"}, {"registryUrl":"https://  
docker.io"}]
```

Important

Es liegt in Ihrer Verantwortung, das `DockerComposeFileDestinationPath`-Verzeichnis zu sichern, in dem Ihre Docker Compose-Datei und die Anmeldeinformationen für Ihre Docker-Abbilder aus privaten Repositorys gespeichert sind. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Parameters

Dieser Konnektor stellt die folgenden Parameter bereit:

Version 7

`DockerComposeFileS3Bucket`

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Achten Sie bei der Erstellung des Buckets darauf, die [Regeln für Bucket-Namen](#) zu befolgen, die im Amazon Simple Storage Service-Benutzerhandbuch beschrieben sind.

Anzeigename in der AWS IoT Konsole: Docker Compose-Datei in S3

Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

Der Objektschlüssel für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie unter [Objektschlüssel und Metadaten](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `.+`

DockerComposeFileS3Version

Die Objektversion für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie [unter Verwenden der Versionierung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `false`

Typ: `string`

Gültiges Muster `.+`

DockerComposeFileDestinationPath

Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern einer Kopie der Docker Compose-Datei verwendet wird. Dies muss ein vorhandenes Verzeichnis sein. Der für

DockerUserId angegebene Benutzer muss über die Berechtigung zum Erstellen einer Datei in diesem Verzeichnis verfügen. Weitere Informationen finden Sie unter [the section called “Einrichten des Docker-Benutzers auf dem Core”](#).

 Important

In diesem Verzeichnis werden Ihre Docker Compose-Datei und die Anmeldeinformationen für Ihre Docker-Abbilder aus privaten Repositorys gespeichert. Es liegt in Ihrer Verantwortung, dieses Verzeichnis zu sichern. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Anzeigename in der AWS IoT Konsole: Verzeichnispfad für die lokale Compose-Datei

Erforderlich: true

Typ: string

Gültiges Muster `\/.*\`

Beispiel: `/home/username/myCompose`

DockerUserId

Die UID des Linux-Benutzers, unter dem der Konnektor ausgeführt wird. Dieser Benutzer muss zur docker-Linux-Gruppe auf dem Core-Gerät gehören und Schreibberechtigungen für das DockerComposeFileDestinationPath-Verzeichnis haben. Weitere Informationen finden Sie unter [Einrichten des Docker-Benutzers auf dem Core](#).

 Note

Wir empfehlen Ihnen, eine Ausführung als Root zu vermeiden, sofern dies nicht absolut notwendig ist. Wenn Sie den Root-Benutzer angeben, müssen Sie zulassen, dass Lambda-Funktionen auf dem AWS IoT Greengrass Core als Root ausgeführt werden. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: false

Typ: string

Gültiges Muster: `^[0-9]{1,5}$`

`AWSecretsArnList`

Die Amazon-Ressourcennamen (ARNs) der darin enthaltenen Geheimnisse enthalten die Anmeldeinformationen AWS Secrets Manager, die für den Zugriff auf Ihre Docker-Images in privaten Repositorys verwendet werden. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositorys”](#).

Anzeigename in der AWS IoT Konsole: Anmeldeinformationen für private Repositorys

Erforderlich: `false`. Dieser Parameter ist erforderlich, um auf Docker-Abbilder zuzugreifen, die in privaten Repositorys gespeichert sind.

Typ: array von string

Gültiges Muster: `[( ? , ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\+\/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+))"]`

`DockerContainerStatusLogFrequency`

Die Häufigkeit (in Sekunden), mit der der Konnektor Statusinformationen über die Docker-Container protokolliert, die auf dem Core ausgeführt werden. Der Standardwert ist 300 Sekunden (5 Minuten).

Anzeigename in der AWS IoT Konsole: Häufigkeit der Protokollierung

Erforderlich: `false`

Typ: string

Gültiges Muster: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Gibt an, ob die Docker-Bereitstellung erzwungen werden soll, wenn sie aufgrund einer unsachgemäßen Bereinigung der letzten Bereitstellung fehlschlägt. Der Standardwert ist `False`.

Anzeigename in der AWS IoT Konsole: Bereitstellung erzwingen

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

`DockerPullBeforeUp`

Gibt an, ob der Deployer ausgeführt werden soll, `docker-compose pull` bevor er `docker-compose up` für ein pull-down-up Verhalten ausgeführt wird. Der Standardwert ist `True`.

Anzeigename in der AWS IoT Konsole: Docker Pull Before Up

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

`StopContainersOnNewDeployment`

Gibt an, ob der Connector die von Docker Deployer verwalteten Docker-Container stoppen soll, wenn GGC gestoppt wird (GGC stoppt, wenn eine neue Gruppe bereitgestellt wird oder der Kernel heruntergefahren wird). Der Standardwert ist `True`.

Anzeigename in der AWS IoT Konsole: Docker stoppt bei neuer Bereitstellung

Note

Wir empfehlen, für diesen Parameter den `True` Standardwert beizubehalten. Der Parameter `to False` bewirkt, dass Ihr Docker-Container auch nach dem Beenden des AWS IoT Greengrass Cores oder dem Starten einer neuen Bereitstellung weiter ausgeführt wird. Wenn Sie diesen Parameter auf `setzen`, müssen Sie sicherstellen `False`, dass Ihre Docker-Container im Falle einer Änderung oder Hinzufügung eines `docker-compose` Servicenamens nach Bedarf beibehalten werden.

Weitere Informationen finden Sie in der Dokumentation zur `docker-compose` Compose-Datei.

Erforderlich: `false`

Typ: string

Gültiges Muster: `^(true|false)$`

DockerOfflineMode

Gibt an, ob die vorhandene Docker Compose-Datei verwendet werden soll, wenn sie offline AWS IoT Greengrass gestartet wird. Der Standardwert ist `False`.

Erforderlich: `false`

Typ: string

Gültiges Muster: `^(true|false)$`

Version 6

DockerComposeFileS3Bucket

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Achten Sie bei der Erstellung des Buckets darauf, die [Regeln für Bucket-Namen](#) zu befolgen, die im Amazon Simple Storage Service-Benutzerhandbuch beschrieben sind.

Anzeigenname in der AWS IoT Konsole: Docker Compose-Datei in S3

Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: string

Gültiges Muster `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

Der Objektschlüssel für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie unter [Objektschlüssel und Metadaten](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `.+`

`DockerComposeFileS3Version`

Die Objektversion für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie [unter Verwenden der Versionierung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `false`

Typ: `string`

Gültiges Muster `.+`

`DockerComposeFileDestinationPath`

Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern einer Kopie der Docker Compose-Datei verwendet wird. Dies muss ein vorhandenes Verzeichnis sein. Der für `DockerUserId` angegebene Benutzer muss über die Berechtigung zum Erstellen einer Datei in diesem Verzeichnis verfügen. Weitere Informationen finden Sie unter [the section called "Einrichten des Docker-Benutzers auf dem Core"](#).

 Important

In diesem Verzeichnis werden Ihre Docker Compose-Datei und die Anmeldeinformationen für Ihre Docker-Abbilder aus privaten Repositorys gespeichert. Es liegt in Ihrer Verantwortung, dieses Verzeichnis zu sichern. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Anzeigename in der AWS IoT Konsole: Verzeichnispfad für die lokale Compose-Datei

Erforderlich: true

Typ: string

Gültiges Muster `\/.*\`

Beispiel: `/home/username/myCompose`

DockerUserId

Die UID des Linux-Benutzers, unter dem der Konnektor ausgeführt wird. Dieser Benutzer muss zur docker-Linux-Gruppe auf dem Core-Gerät gehören und Schreibberechtigungen für das DockerComposeFileDestinationPath-Verzeichnis haben. Weitere Informationen finden Sie unter [Einrichten des Docker-Benutzers auf dem Core](#).

 Note

Wir empfehlen Ihnen, eine Ausführung als Root zu vermeiden, sofern dies nicht absolut notwendig ist. Wenn Sie den Root-Benutzer angeben, müssen Sie zulassen, dass Lambda-Funktionen auf dem AWS IoT Greengrass Core als Root ausgeführt werden. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: false

Typ: string

Gültiges Muster: `^[0-9]{1,5}`

AWSecretsArnList

Die Amazon-Ressourcennamen (ARNs) der darin enthaltenen Geheimnisse enthalten die Anmeldeinformationen AWS Secrets Manager, die für den Zugriff auf Ihre Docker-Images in privaten Repositorys verwendet werden. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositorys”](#).

Anzeigename in der AWS IoT Konsole: Anmeldeinformationen für private Repositorys

Erforderlich: `false`. Dieser Parameter ist erforderlich, um auf Docker-Abbilder zuzugreifen, die in privaten Repositorys gespeichert sind.

Typ: `array von string`

Gültiges Muster: `[( ? , ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+)/[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)"]`

DockerContainerStatusLogFrequency

Die Häufigkeit (in Sekunden), mit der der Konnektor Statusinformationen über die Docker-Container protokolliert, die auf dem Core ausgeführt werden. Der Standardwert ist 300 Sekunden (5 Minuten).

Anzeigename in der AWS IoT Konsole: Häufigkeit der Protokollierung

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

Gibt an, ob die Docker-Bereitstellung erzwungen werden soll, wenn sie aufgrund einer unsachgemäßen Bereinigung der letzten Bereitstellung fehlschlägt. Der Standardwert ist `False`.

Anzeigename in der AWS IoT Konsole: Bereitstellung erzwingen

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

DockerPullBeforeUp

Gibt an, ob der Deployer ausgeführt werden soll, `docker-compose pull` bevor er `docker-compose up` für ein pull-down-up Verhalten ausgeführt wird. Der Standardwert ist `True`.

Anzeigename in der AWS IoT Konsole: Docker Pull Before Up

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

StopContainersOnNewDeployment

Gibt an, ob der Connector die von Docker Deployer verwalteten Docker-Container stoppen soll, wenn GGC gestoppt wird (wenn eine neue Gruppenbereitstellung erfolgt oder der Kernel heruntergefahren wird). Der Standardwert ist `True`.

Anzeigename in der AWS IoT Konsole: Docker stoppt bei neuer Bereitstellung

Note

Wir empfehlen, für diesen Parameter den `True` Standardwert beizubehalten. Der Parameter `to False` bewirkt, dass Ihr Docker-Container auch nach dem Beenden des AWS IoT Greengrass Cores oder dem Starten einer neuen Bereitstellung weiter ausgeführt wird. Wenn Sie diesen Parameter auf `setzen`, müssen Sie sicherstellen `False`, dass Ihre Docker-Container im Falle einer Änderung oder Hinzufügung eines `docker-compose` Servicenamens nach Bedarf beibehalten werden.

Weitere Informationen finden Sie in der Dokumentation zur `docker-compose` Compose-Datei.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^(true|false)$`

Version 5

DockerComposeFileS3Bucket

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Achten Sie bei der Erstellung des Buckets darauf, die [Regeln für Bucket-Namen](#) zu befolgen, die im Amazon Simple Storage Service-Benutzerhandbuch beschrieben sind.

Anzeigename in der AWS IoT Konsole: Docker Compose-Datei in S3

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `[a-zA-Z0-9\\-\\.]{3,63}`

DockerComposeFileS3Key

Der Objektschlüssel für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie unter [Objektschlüssel und Metadaten](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `.+`

DockerComposeFileS3Version

Die Objektversion für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie [unter Verwenden der Versionierung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: false

Typ: string

Gültiges Muster .+

DockerComposeFileDestinationPath

Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern einer Kopie der Docker Compose-Datei verwendet wird. Dies muss ein vorhandenes Verzeichnis sein. Der für `DockerUserId` angegebene Benutzer muss über die Berechtigung zum Erstellen einer Datei in diesem Verzeichnis verfügen. Weitere Informationen finden Sie unter [the section called “Einrichten des Docker-Benutzers auf dem Core”](#).

Important

In diesem Verzeichnis werden Ihre Docker Compose-Datei und die Anmeldeinformationen für Ihre Docker-Abbilder aus privaten Repositorys gespeichert. Es liegt in Ihrer Verantwortung, dieses Verzeichnis zu sichern. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Anzeigenname in der AWS IoT Konsole: Verzeichnispfad für die lokale Compose-Datei

Erforderlich: true

Typ: string

Gültiges Muster `\/.*\`

Beispiel: `/home/username/myCompose`

DockerUserId

Die UID des Linux-Benutzers, unter dem der Konnektor ausgeführt wird. Dieser Benutzer muss zur docker-Linux-Gruppe auf dem Core-Gerät gehören und Schreibberechtigungen für das DockerComposeFileDestinationPath-Verzeichnis haben. Weitere Informationen finden Sie unter [Einrichten des Docker-Benutzers auf dem Core](#).

Note

Wir empfehlen Ihnen, eine Ausführung als Root zu vermeiden, sofern dies nicht absolut notwendig ist. Wenn Sie den Root-Benutzer angeben, müssen Sie zulassen, dass Lambda-Funktionen auf dem AWS IoT Greengrass Core als Root ausgeführt werden. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}$`

AWSecretsArnList

Die Amazon-Ressourcennamen (ARNs) der darin enthaltenen Geheimnisse enthalten die Anmeldeinformationen AWS Secrets Manager, die für den Zugriff auf Ihre Docker-Images in privaten Repositories verwendet werden. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositories”](#).

Anzeigename in der AWS IoT Konsole: Anmeldeinformationen für private Repositories

Erforderlich: `false`. Dieser Parameter ist erforderlich, um auf Docker-Abbilder zuzugreifen, die in privaten Repositories gespeichert sind.

Typ: `array von string`

Gültiges Muster: `[ ( ? , ? ? " ( arn : ( aws ( - [ a - z ] + ) ) : secretsmanager : [ a - z 0 - 9 - ] + : [ 0 - 9 ] { 12 } : secret : ( [ a - z A - Z 0 - 9 \ ] + / ) [ a - z A - Z 0 - 9 / _ + = , . @ - ] + - [ a - z A - Z 0 - 9 ] + ) " ) ]`

DockerContainerStatusLogFrequency

Die Häufigkeit (in Sekunden), mit der der Konnektor Statusinformationen über die Docker-Container protokolliert, die auf dem Core ausgeführt werden. Der Standardwert ist 300 Sekunden (5 Minuten).

Anzeigename in der AWS IoT Konsole: Häufigkeit der Protokollierung

Erforderlich: false

Typ: string

Gültiges Muster: `^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

Gibt an, ob die Docker-Bereitstellung erzwungen werden soll, wenn sie aufgrund einer unsachgemäßen Bereinigung der letzten Bereitstellung fehlschlägt. Der Standardwert ist False.

Anzeigename in der AWS IoT Konsole: Bereitstellung erzwingen

Erforderlich: false

Typ: string

Gültiges Muster: `^(true|false)$`

DockerPullBeforeUp

Gibt an, ob der Deployer ausgeführt werden soll, `docker-compose pull` bevor er `docker-compose up` für ein pull-down-up Verhalten ausgeführt wird. Der Standardwert ist True.

Anzeigename in der AWS IoT Konsole: Docker Pull Before Up

Erforderlich: false

Typ: string

Gültiges Muster: `^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Achten Sie bei der Erstellung des Buckets darauf, die [Regeln für Bucket-Namen](#) zu befolgen, die im Amazon Simple Storage Service-Benutzerhandbuch beschrieben sind.

Anzeigename in der AWS IoT Konsole: Docker Compose-Datei in S3

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Der Objektschlüssel für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie unter [Objektschlüssel und Metadaten](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `.+`

DockerComposeFileS3Version

Die Objektversion für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie [unter Verwenden der Versionierung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: false

Typ: string

Gültiges Muster .+

DockerComposeFileDestinationPath

Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern einer Kopie der Docker Compose-Datei verwendet wird. Dies muss ein vorhandenes Verzeichnis sein. Der für `DockerUserId` angegebene Benutzer muss über die Berechtigung zum Erstellen einer Datei in diesem Verzeichnis verfügen. Weitere Informationen finden Sie unter [the section called “Einrichten des Docker-Benutzers auf dem Core”](#).

Important

In diesem Verzeichnis werden Ihre Docker Compose-Datei und die Anmeldeinformationen für Ihre Docker-Abbilder aus privaten Repositorys gespeichert. Es liegt in Ihrer Verantwortung, dieses Verzeichnis zu sichern. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Anzeigenname in der AWS IoT Konsole: Verzeichnispfad für die lokale Compose-Datei

Erforderlich: true

Typ: string

Gültiges Muster `\/.*\`

Beispiel: `/home/username/myCompose`

DockerUserId

Die UID des Linux-Benutzers, unter dem der Konnektor ausgeführt wird. Dieser Benutzer muss zur docker-Linux-Gruppe auf dem Core-Gerät gehören und Schreibberechtigungen für das DockerComposeFileDestinationPath-Verzeichnis haben. Weitere Informationen finden Sie unter [Einrichten des Docker-Benutzers auf dem Core](#).

Note

Wir empfehlen Ihnen, eine Ausführung als Root zu vermeiden, sofern dies nicht absolut notwendig ist. Wenn Sie den Root-Benutzer angeben, müssen Sie zulassen, dass Lambda-Funktionen auf dem AWS IoT Greengrass Core als Root ausgeführt werden. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[0-9]{1,5}`

AWSecretsArnList

Die Amazon-Ressourcennamen (ARNs) der darin enthaltenen Geheimnisse enthalten die Anmeldeinformationen AWS Secrets Manager, die für den Zugriff auf Ihre Docker-Images in privaten Repositories verwendet werden. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositories”](#).

Anzeigename in der AWS IoT Konsole: Anmeldeinformationen für private Repositories

Erforderlich: `false`. Dieser Parameter ist erforderlich, um auf Docker-Abbilder zuzugreifen, die in privaten Repositories gespeichert sind.

Typ: `array von string`

Gültiges Muster: `[( ? , ? ? "(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)"]`

DockerContainerStatusLogFrequency

Die Häufigkeit (in Sekunden), mit der der Konnektor Statusinformationen über die Docker-Container protokolliert, die auf dem Core ausgeführt werden. Der Standardwert ist 300 Sekunden (5 Minuten).

Anzeigename in der AWS IoT Konsole: Häufigkeit der Protokollierung

Erforderlich: false

Typ: string

Gültiges Muster: `^[1-9]{1}[0-9]{0,3}$`

ForceDeploy

Gibt an, ob die Docker-Bereitstellung erzwungen werden soll, wenn sie aufgrund einer unsachgemäßen Bereinigung der letzten Bereitstellung fehlschlägt. Der Standardwert ist False.

Anzeigename in der AWS IoT Konsole: Bereitstellung erzwingen

Erforderlich: false

Typ: string

Gültiges Muster: `^(true|false)$`

Version 1

DockerComposeFileS3Bucket

Der Name des S3-Buckets, der Ihre Docker Compose-Datei enthält. Achten Sie bei der Erstellung des Buckets darauf, die [Regeln für Bucket-Namen](#) zu befolgen, die im Amazon Simple Storage Service-Benutzerhandbuch beschrieben sind.

Anzeigename in der AWS IoT Konsole: Docker Compose-Datei in S3

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Der Objektschlüssel für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie unter [Objektschlüssel und Metadaten](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster `.+`

`DockerComposeFileS3Version`

Die Objektversion für Ihre Docker Compose-Datei in Amazon S3. Weitere Informationen, einschließlich Richtlinien zur Benennung von Objektschlüsseln, finden Sie [unter Verwenden der Versionierung](#) im Amazon Simple Storage Service-Benutzerhandbuch.

 Note

In der Konsole kombiniert die Docker Compose-Datei in S3-Eigenschaft die Parameter `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` und `DockerComposeFileS3Version`.

Erforderlich: `false`

Typ: `string`

Gültiges Muster `.+`

`DockerComposeFileDestinationPath`

Der absolute Pfad des lokalen Verzeichnisses, das zum Speichern einer Kopie der Docker Compose-Datei verwendet wird. Dies muss ein vorhandenes Verzeichnis sein. Der für `DockerUserId` angegebene Benutzer muss über die Berechtigung zum Erstellen einer Datei in diesem Verzeichnis verfügen. Weitere Informationen finden Sie unter [the section called “Einrichten des Docker-Benutzers auf dem Core”](#).

 Important

In diesem Verzeichnis werden Ihre Docker Compose-Datei und die Anmeldeinformationen für Ihre Docker-Abbilder aus privaten Repositories gespeichert. Es liegt in Ihrer Verantwortung, dieses Verzeichnis zu sichern. Weitere Informationen finden Sie unter [the section called “Sicherheitshinweise”](#).

Anzeigename in der AWS IoT Konsole: Verzeichnispfad für die lokale Compose-Datei

Erforderlich: `true`

Typ: `string`

Gültiges Muster `\. * \/?`

Beispiel: `/home/username/myCompose`

DockerUserId

Die UID des Linux-Benutzers, unter dem der Konnektor ausgeführt wird. Dieser Benutzer muss zur docker-Linux-Gruppe auf dem Core-Gerät gehören und Schreibberechtigungen für das DockerComposeFileDestinationPath-Verzeichnis haben. Weitere Informationen finden Sie unter [Einrichten des Docker-Benutzers auf dem Core](#).

Note

Wir empfehlen Ihnen, eine Ausführung als Root zu vermeiden, sofern dies nicht absolut notwendig ist. Wenn Sie den Root-Benutzer angeben, müssen Sie zulassen, dass Lambda-Funktionen auf dem AWS IoT Greengrass Core als Root ausgeführt werden. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Anzeigename in der AWS IoT Konsole: Docker-Benutzer-ID

Erforderlich: false

Typ: string

Gültiges Muster: `^[0-9]{1,5}$`

AWSecretsArnList

Die Amazon-Ressourcennamen (ARNs) der darin enthaltenen Geheimnisse enthalten die Anmeldeinformationen AWS Secrets Manager, die für den Zugriff auf Ihre Docker-Images in privaten Repositorys verwendet werden. Weitere Informationen finden Sie unter [the section called “Zugreifen auf Docker-Images aus privaten Repositorys”](#).

Anzeigename in der AWS IoT Konsole: Anmeldeinformationen für private Repositorys

Erforderlich: false. Dieser Parameter ist erforderlich, um auf Docker-Abbilder zuzugreifen, die in privaten Repositorys gespeichert sind.

Typ: array von string

Gültiges Muster: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+)")]`

DockerContainerStatusLogFrequency

Die Häufigkeit (in Sekunden), mit der der Konnektor Statusinformationen über die Docker-Container protokolliert, die auf dem Core ausgeführt werden. Der Standardwert ist 300 Sekunden (5 Minuten).

Anzeigenname in der AWS IoT Konsole: Häufigkeit der Protokollierung

Erforderlich: false

Typ: string

Gültiges Muster: `^[1-9]{1}[0-9]{0,3}$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt einen ConnectorDefinition mit einer ersten Version, die den Greengrass Docker Application Deployment Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyDockerAppplicationDeploymentConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
DockerApplicationDeployment/versions/5",
      "Parameters": {
        "DockerComposeFileS3Bucket": "amzn-s3-demo-bucket",
        "DockerComposeFileS3Key": "production-docker-compose.yml",
        "DockerComposeFileS3Version": "123",
        "DockerComposeFileDestinationPath": "/home/username/myCompose",
        "DockerUserId": "1000",
        "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\", \"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
        "DockerContainerStatusLogFrequency": "30",
        "ForceDeploy": "True",
        "DockerPullBeforeUp": "True"
      }
    }
  ]
}'
```

 Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

Eingabedaten

Dieser Konnektor benötigt oder akzeptiert keine Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht den Status des `docker-compose up`-Befehls als Ausgabedaten.

Themenfilter im Abonnement

`dockerapplicationdeploymentconnector/message/status`

Beispielausgabe: Erfolg

```
{
  "status": "success",
  "GreengrassDockerApplicationDeploymentStatus": "Successfully triggered docker-
compose up",
  "S3Bucket": "amzn-s3-demo-bucket",
  "ComposeFileName": "production-docker-compose.yml",
  "ComposeFileVersion": "123"
}
```

Beispielausgabe: Fehler

```
{
  "status": "fail",
  "error_message": "description of error",
  "error": "InvalidParameter"
}
```

Der Fehlertyp kann `InvalidParameter` oder `InternalError` sein.

Den Docker-Benutzer auf dem Core einrichten AWS IoT Greengrass

Der Greengrass Docker Application Deployment Connector wird als der Benutzer ausgeführt, den Sie für den `DockerUserId` Parameter angeben. Wenn Sie keinen Wert angeben, wird der Konnektor als `ggc_user` ausgeführt. Dies ist die standardmäßige Greengrass-Zugriffsidentität.

Damit der Konnektor mit dem Docker-Daemon interagieren kann, muss der Docker-Benutzer zur `docker-Linux`-Gruppe auf dem Core gehören. Der Docker-Benutzer muss auch über Schreibberechtigungen für das `DockerComposeFileDestinationPath`-Verzeichnis verfügen. Hier speichert der Konnektor Ihre lokale `docker-compose.yml`-Datei und die Docker-Anmeldeinformationen.

Note

- Es wird empfohlen, einen Linux-Benutzer zu erstellen, anstatt den Standard-`ggc_user` zu verwenden. Andernfalls kann jede Lambda-Funktion in der Greengrass-Gruppe auf die Compose-Datei und die Docker-Anmeldeinformationen zugreifen.
- Wir empfehlen Ihnen, eine Ausführung als Root zu vermeiden, sofern dies nicht absolut notwendig ist. Wenn Sie den Root-Benutzer angeben, müssen Sie zulassen, dass Lambda-Funktionen auf dem AWS IoT Greengrass Core als Root ausgeführt werden. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

1. Erstellen Sie den Benutzer. Sie können den `useradd`-Befehl ausführen und die optionale `-u`-Option zum Zuweisen einer UID hinzufügen. Beispiel:

```
sudo useradd -u 1234 user-name
```

2. Fügen Sie den Benutzer der `docker`-Gruppe auf dem Core hinzu. Beispiel:

```
sudo usermod -aG docker user-name
```

Weitere Informationen, einschließlich dem Erstellen der `docker`-Gruppe, finden Sie unter [Verwalten von Docker als Nicht-Root-Benutzer](#) in der Docker-Dokumentation.

3. Erteilen Sie dem Benutzer die Berechtigungen, in das für den `DockerComposeFileDestinationPath`-Parameter angegebene Verzeichnis zu schreiben.

Beispiel:

- a. So legen Sie den Benutzer als Besitzer des Verzeichnisses fest. In diesem Beispiel wird die UID aus Schritt 1 verwendet.

```
chown 1234 docker-compose-file-destination-path
```

- b. So erteilen Sie dem Besitzer Lese- und Schreibberechtigungen.

```
chmod 700 docker-compose-file-destination-path
```

Weitere Informationen finden Sie unter [How To Manage File And Folder Permissions In Linux](#) in der Linux Foundation-Dokumentation.

- c. Wenn Sie beim Erstellen des Benutzers keine UID zugewiesen haben oder wenn Sie einen vorhandenen Benutzer verwendet haben, führen Sie den `id`-Befehl aus, um die UID zu suchen.

```
id -u user-name
```

Sie verwenden die UID, um den `DockerUserId`-Parameter für den Konnektor zu konfigurieren.

Informationen zur Nutzung

Wenn Sie den Greengrass Docker Application Deployment Connector verwenden, sollten Sie die folgenden implementierungsspezifischen Nutzungsinformationen beachten.

- Das Präfix für Projektnamen wurde korrigiert. Der Konnektor stellt das `greengrassdockerapplicationdeployment`-Präfix den Namen der Docker-Container vor, die er startet. Der Konnektor verwendet dieses Präfix als Projektname in den ausgeführten `docker-compose`-Befehlen.
- Verhalten beim Protokollieren. Der Konnektor schreibt Statusinformationen und Fehlerbehebungsinformationen in eine Protokolldatei. Sie können so konfigurieren AWS IoT Greengrass, dass CloudWatch Protokolle an Logs gesendet und Logs lokal geschrieben werden.

Weitere Informationen finden Sie unter [the section called “Protokollierung”](#). Dies ist der Pfad zum lokalen Protokoll für den Konnektor:

```
/greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
```

Sie müssen über Root-Berechtigungen verfügen, um auf lokale Protokolle zugreifen zu können.

- Docker-Images werden aktualisiert. Docker speichert Abbilder auf dem Core-Gerät zwischen. Wenn Sie ein Docker-Abbild aktualisieren und die Änderung an das Core-Gerät weitergeben möchten, stellen Sie sicher, dass Sie das Tag für das Abbild in der Compose-Datei ändern. Änderungen werden wirksam, nachdem die Greengrass-Gruppe bereitgestellt wurde.
- 10-minütiges Timeout für Aufräumarbeiten. Wenn der Greengrass-Daemon während eines Neustarts stoppt, wird der `docker-compose down` Befehl initiiert. Alle Docker-Container haben nach der Initiierung maximal 10 Minuten Zeit, um alle `docker-compose down` Bereinigungsvorgänge durchzuführen. Wenn die Bereinigung nicht innerhalb von 10 Minuten abgeschlossen ist, müssen Sie die verbleibenden Container manuell bereinigen. Weitere Informationen finden Sie unter [docker rm](#) in der Docker-CLI-Dokumentation.
- Docker-Befehle ausführen. Um Probleme zu beheben, können Sie Docker-Befehle in einem Terminalfenster auf dem Core-Gerät ausführen. Führen Sie beispielsweise den folgenden Befehl aus, um die Docker-Container anzuzeigen, die vom Konnektor gestartet wurden:

```
docker ps --filter name="greengrassdockerapplicationdeployment"
```

- Reservierte Ressourcen-ID. Der Konnektor verwendet die `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_`*index*-ID für die Greengrass-Ressourcen, die er in der Greengrass-Gruppe erstellt. Die Ressource IDs muss in der Gruppe eindeutig sein. Weisen Sie daher keine Ressourcen-ID zu, die mit dieser reservierten Ressourcen-ID in Konflikt geraten könnte.
- Offline-Modus. Wenn Sie den `DockerOfflineMode` Konfigurationsparameter auf `setzenTrue`, kann der Docker-Connector im Offline-Modus betrieben werden. Dies kann passieren, wenn eine Greengrass-Gruppenbereitstellung neu gestartet wird, während das Kerngerät offline ist und der Connector keine Verbindung zu Amazon S3 oder Amazon ECR herstellen kann, um die Docker Compose-Datei abzurufen.

Wenn der Offline-Modus aktiviert ist, versucht der Connector, Ihre Compose-Datei herunterzuladen und `docker login` Befehle auszuführen, wie dies bei einem normalen Neustart der Fall wäre. Wenn diese Versuche fehlschlagen, sucht der Connector nach einer lokal gespeicherten

Compose-Datei in dem Ordner, der mit dem `DockerComposeFileDestinationPath` Parameter angegeben wurde. Wenn eine lokale Compose-Datei vorhanden ist, folgt der Konnektor der normalen docker-compose Befehlsfolge und bezieht Daten aus lokalen Bildern. Wenn die Compose-Datei oder die lokalen Bilder nicht vorhanden sind, schlägt der Connector fehl. Das Verhalten der `StopContainersOnNewDeployment` Parameter `ForceDeploy` und bleibt im Offline-Modus gleich.

Kommunikation mit Docker-Containern

AWS IoT Greengrass unterstützt die folgenden Kommunikationskanäle zwischen Greengrass-Komponenten und Docker-Containern:

- Greengrass Lambda-Funktionen können REST verwenden, APIs um mit Prozessen in Docker-Containern zu kommunizieren. Sie können einen Server in einem Docker-Container einrichten, der einen Port öffnet. Lambda-Funktionen können mit dem Container an diesem Port kommunizieren.
- Prozesse in Docker-Containern können MQTT-Nachrichten über den lokalen Greengrass-Nachrichtenbroker austauschen. Sie können den Docker-Container als Client-Gerät in der Greengrass-Gruppe einrichten und dann Abonnements erstellen, damit der Container mit Greengrass Lambda-Funktionen, Client-Geräten und anderen Connectoren in der Gruppe oder mit AWS IoT und dem lokalen Shadow-Service kommunizieren kann. Weitere Informationen finden Sie unter [the section called “Konfigurieren der MQTT-Kommunikation mit Docker-Containern”](#).
- Die Lambda-Funktionen von Greengrass können eine gemeinsam genutzte Datei aktualisieren, um Informationen an Docker-Container weiterzuleiten. Sie können die Compose-Datei für ein Bind-Mount des freigegebenen Dateipfads für einen Docker-Container verwenden.

Konfigurieren der MQTT-Kommunikation mit Docker-Containern

Sie können einen Docker-Container als Client-Gerät konfigurieren und ihn einer Greengrass-Gruppe hinzufügen. Anschließend können Sie Abonnements erstellen, die die MQTT-Kommunikation zwischen dem Docker-Container und Greengrass-Komponenten oder AWS IoT ermöglichen. Im folgenden Verfahren erstellen Sie ein Abonnement, mit dem das Docker-Containergerät Schattenaktualisierungsmeldungen vom lokalen Schattenservice empfangen kann. Sie können diesem Muster folgen, um andere Abonnements zu erstellen.

 Note

Bei diesem Verfahren wird davon ausgegangen, dass Sie bereits eine Greengrass-Gruppe und einen Greengrass-Kern (v1.10 oder höher) erstellt haben. Informationen zum Erstellen einer Greengrass-Gruppe und eines Greengrass-Kerns finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#).

Um einen Docker-Container als Client-Gerät zu konfigurieren und ihn einer Greengrass-Gruppe hinzuzufügen

1. Erstellen Sie einen Ordner auf dem Core-Gerät, um die Zertifikate und Schlüssel zu speichern, die zur Authentifizierung des Greengrass-Geräts verwendet werden.

Der Dateipfad muss auf dem Docker-Container gemountet werden, den Sie starten möchten. Das folgende Snippet zeigt, wie Sie einen Dateipfad in Ihrer Compose-Datei mounten. Stellt in diesem Beispiel den Ordner *path-to-device-certs* dar, den Sie in diesem Schritt erstellt haben.

```
version: '3.3'
services:
  myService:
    image: user-name/repo:image-tag
    volumes:
      - /path-to-device-certs/:/path-accessible-in-container
```

2. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
3. Wählen Sie die Zielgruppe aus.
4. Wählen Sie auf der Gruppenkonfigurationsseite die Option Client-Geräte und dann Associate aus.
5. Wählen Sie im Modal Ein Client-Gerät mit dieser Gruppe verknüpfen die Option Neues Objekt erstellen AWS IoT aus.

Die Seite „Dinge erstellen“ wird auf einer neuen Registerkarte geöffnet.

6. Wählen Sie auf der Seite „Dinge erstellen“ die Option „Einzelnes Ding erstellen“ und dann „Weiter“ aus.

7. Geben Sie auf der Seite „Eigenschaften des Objekts angeben“ einen Namen für das Gerät ein und wählen Sie dann Weiter aus.
8. Wählen Sie auf der Seite Gerätezertifikat konfigurieren die Option Weiter aus.
9. Führen Sie auf der Seite Richtlinien an Zertifikat anhängen einen der folgenden Schritte aus:
 - Wählen Sie eine bestehende Richtlinie aus, die Berechtigungen gewährt, die Client-Geräte benötigen, und wählen Sie dann Ding erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit dem Core AWS Cloud und dem Core verwendet.

- Erstellen Sie eine neue Richtlinie, die Client-Geräteberechtigungen gewährt, und fügen Sie sie an. Gehen Sie wie folgt vor:
 - a. Wählen Sie Richtlinie erstellen aus.

Die Seite Create policy (Richtlinie erstellen) wird in einer neuen Registerkarte geöffnet.

- b. Führen Sie auf der Seite Create policy (Richtlinie erstellen) die folgenden Schritte aus:
 - i. Geben Sie als Richtlinienname einen Namen ein, der die Richtlinie beschreibt, z. **GreengrassV1ClientDevicePolicy B**.
 - ii. Wählen Sie auf der Registerkarte Richtlinienenerklärungen unter Richtlinienokument die Option JSON aus.
 - iii. Geben Sie das folgende Richtlinienokument ein. Diese Richtlinie ermöglicht es dem Client-Gerät, Greengrass-Kerne zu erkennen und zu allen MQTT-Themen zu kommunizieren. Informationen darüber, wie Sie den Zugriff dieser Richtlinie einschränken können, finden Sie unter. [Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass](#)

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
```

```
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

- iv. Wählen Sie Create (Erstellen) aus, um die Richtlinie zu erstellen.
- c. Kehren Sie zur Browser-Registerkarte zurück, auf der die Seite Richtlinien an Zertifikat anhängen geöffnet ist. Gehen Sie wie folgt vor:
 - i. Wählen Sie in der Liste Richtlinien die Richtlinie aus, die Sie erstellt haben, z. GreengrassV1ClientDevicePolicyB.

Wenn Sie die Richtlinie nicht sehen, klicken Sie auf die Schaltfläche „Aktualisieren“.

- ii. Wählen Sie Objekt erstellen aus.

Es öffnet sich ein Fenster, in dem Sie die Zertifikate und Schlüssel herunterladen können, die das Gerät für die Verbindung mit dem Core AWS Cloud und dem Core verwendet.

- 10. Laden Sie im Modal Zertifikate und Schlüssel herunterladen die Zertifikate des Geräts herunter.

 Important

Laden Sie die Sicherheitsressourcen herunter, bevor Sie „Fertig“ wählen.

Gehen Sie wie folgt vor:

- a. Wählen Sie unter Gerätezertifikat die Option Herunterladen aus, um das Gerätezertifikat herunterzuladen.
- b. Wählen Sie unter Datei mit öffentlichem Schlüssel die Option Herunterladen aus, um den öffentlichen Schlüssel für das Zertifikat herunterzuladen.
- c. Wählen Sie unter Datei mit privatem Schlüssel die Option Herunterladen aus, um die Datei mit dem privaten Schlüssel für das Zertifikat herunterzuladen.
- d. Lesen Sie die [Serverauthentifizierung](#) im AWS IoT Entwicklerhandbuch und wählen Sie das entsprechende Root-CA-Zertifikat aus. Wir empfehlen die Verwendung von Amazon Trust Services (ATS) -Endpunkten und ATS-Root-CA-Zertifikaten. Wählen Sie unter Root-CA-Zertifikate die Option Herunterladen für ein Root-CA-Zertifikat aus.
- e. Wählen Sie Fertig aus.

Notieren Sie sich die Zertifikat-ID, die in den Dateinamen für das Gerätezertifikat und die Schlüssel häufig vorkommt. Sie benötigen sie später.

11. Kopieren Sie die Zertifikate und Schlüssel in den Ordner, den Sie in Schritt 1 erstellt haben.

Erstellen Sie als Nächstes ein Abonnement in der Gruppe. In diesem Beispiel können Sie ein Abonnement erstellen, damit das Docker-Containergerät MQTT-Nachrichten vom lokalen Schattenservice empfangen kann.

 Note

Die maximale Größe eines Schattendokuments beträgt 8 KB. Weitere Informationen finden Sie unter [AWS IoT -Kontingente](#) im AWS IoT -Entwicklerhandbuch.

So erstellen Sie ein Abonnement, mit dem das Docker-Containergerät MQTT-Nachrichten vom lokalen Schattenservice empfangen kann

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.
2. Konfigurieren Sie auf der Seite Select your source and target die Quelle und das Ziel wie folgt:
 - a. Wählen Sie für Eine Quelle auswählen die Option Services und danach Local Shadow Service (Service lokaler Schatten) aus.

- b. Wählen Sie für Ziel auswählen die Option Geräte und dann Ihr Gerät aus.
- c. Wählen Sie Weiter aus.
- d. Wählen Sie auf der Seite Ihre Daten mit einem Thema filtern für Themenfilter die Option **\$aws/things/MyDockerDevice/shadow/update/accepted** und dann Weiter aus. **MyDockerDevice** Ersetzen Sie es durch den Namen des Geräts, das Sie zuvor erstellt haben.
- e. Wählen Sie Finish (Abschließen).

Fügen Sie den folgenden Codeausschnitt in das Docker-Abbild ein, das Sie in der Compose-Datei referenzieren. Das ist der Greengrass-Gerätecode. Fügen Sie außerdem Code in Ihrem Docker-Container hinzu, der das Greengrass-Gerät innerhalb des Containers startet. Es kann als separater Prozess im Abbild oder in einem separaten Thread ausgeführt werden.

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker
  container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"

# Discover Greengrass cores.
```

```
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10) # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None

try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
    coreList = discoveryInfo.getAllCores()

    # Use first discovery result.
    groupId, ca = caList[0]
    coreInfo = coreList[0]

    # Save the group CA to a local file.
    groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
    if not os.path.exists(GROUP_CA_PATH):
        os.makedirs(GROUP_CA_PATH)
    groupCAFile = open(groupCA, "w")
    groupCAFile.write(ca)
    groupCAFile.close()
    discovered = True
except DiscoveryInvalidRequestException as e:
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)
```

```
# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
    if connected:
        break

if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
    time.sleep(1)
```

Sicherheitshinweise

Wenn Sie den Greengrass Docker Application Deployment Connector verwenden, sollten Sie die folgenden Sicherheitsaspekte beachten.

Lokale Speicherung der Docker Compose-Datei

Der Konnektor speichert eine Kopie der Compose-Datei in dem für den `DockerComposeFileDestinationPath`-Parameter angegebenen Verzeichnis.

Es liegt in Ihrer Verantwortung, dieses Verzeichnis zu sichern. Sie sollten Dateisystemberechtigungen verwenden, um den Zugriff auf das Verzeichnis zu beschränken.

Lokale Speicherung der Docker-Anmeldeinformationen

Wenn Ihre Docker-Abbilder in privaten Repositorys gespeichert sind, speichert der Konnektor Ihre Docker-Anmeldeinformationen in dem für den `DockerComposeFileDestinationPath`-Parameter angegebenen Verzeichnis.

Es liegt in Ihrer Verantwortung, diese Anmeldeinformationen zu sichern. Beispielsweise sollten Sie [credential-helper](#) auf dem Core-Gerät verwenden, wenn Sie Docker Engine installieren.

Installieren von Docker Engine von einer vertrauenswürdigen Quelle

Es liegt in Ihrer Verantwortung, Docker Engine von einer vertrauenswürdigen Quelle zu installieren. Dieser Konnektor verwendet den Docker-Daemon auf dem Core-Gerät, um auf Ihre Docker-Komponenten zuzugreifen und Docker-Container zu verwalten.

Umfang der Greengrass-Gruppenrollenberechtigungen

Berechtigungen, die Sie in der Greengrass-Gruppenrolle hinzufügen, können von allen Lambda-Funktionen und -Konnektoren in der Greengrass-Gruppe übernommen werden. Dieser Konnektor erfordert Zugriff auf Ihre Docker Compose-Datei, die in einem S3-Bucket gespeichert ist. Außerdem ist Zugriff auf Ihr Amazon ECR-Autorisierungstoken erforderlich, wenn Ihre Docker-Images in einem privaten Repository in Amazon ECR gespeichert sind.

Lizenzen

Der Greengrass Docker Application Deployment Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0

- [urllib3/MIT-Lizenz](#)

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
7	Es wurde hinzugefügt <code>DockerOfflineMode</code> , um eine vorhandene Docker Compose-Datei zu verwenden, wenn sie offline AWS IoT Greengrass gestartet wird. Wiederholungsversuche für den <code>docker login</code> Befehl wurden implementiert. Support für 32-Bit UUIDs.
6	Es wurde hinzugefügt <code>StopContainersOnNewDeployment</code> , um die Container-Bereinigung zu überschreiben, wenn eine neue Bereitstellung erfolgt oder GGC beendet wird. Sicherere Mechanismen zum Herunterfahren und Starten. Fehlerbehebung bei der YAML-Validierung.
5	Bilder werden vor der Ausführung <code>docker-compose down</code> abgerufen.
4	<code>pull-before-up</code> Verhalten zum Aktualisieren von Docker-Images hinzugefügt.
3	Es wurde ein Problem beim Suchen von Umgebungsvariablen behoben.
2	Der Parameter <code>ForceDeploy</code> wurde hinzugefügt.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)

- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

IoT-Analytics-Konnektor

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und AWS IoT Greengrass wird keine Updates veröffentlichen, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der IoT Analytics Analytics-Connector sendet lokale Gerätedaten an AWS IoT Analytics. Sie können diesen Anschluss als zentralen Hub verwenden, um Daten von Sensoren auf dem Greengrass-Core-Gerät und von [verbundenen Client-Geräten](#) zu sammeln. Der Connector sendet die Daten an AWS IoT Analytics Kanäle in der aktuellen Region AWS-Konto und in der Region. Sie kann Daten an einen Standardzielkanal und an dynamisch angegebene Kanäle senden.

Note

AWS IoT Analytics ist ein vollständig verwalteter Service, mit dem Sie IoT-Daten sammeln, speichern, verarbeiten und abfragen können. Darin AWS IoT Analytics können die Daten weiter analysiert und verarbeitet werden. Auf diese Weise können beispielsweise ML-Modelle zur Überwachung des Computerstatus oder zum Test neuer Modellierungsstrategien geschult werden. Weitere Informationen finden Sie unter [Was ist AWS IoT Analytics?](#) im AWS IoT Analytics Benutzerhandbuch.

Der Konnektor akzeptiert formatierte und unformatierte Daten zu [Eingabe-MQTT-Themen](#). Es werden zwei vordefinierte Themen mit angegebenem Zielkanal unterstützt. Sie kann auch Nachrichten zu den von Kunden definierten Themen empfangen, die [in Abonnements konfiguriert](#) sind. Dies kann verwendet werden, um Nachrichten von Client-Geräten weiterzuleiten, die zu festen Themen veröffentlichen oder unstrukturierte oder stapelabhängige Daten von Geräten mit eingeschränkten Ressourcen verarbeiten.

Dieser Connector verwendet die [BatchPutMessageAPI](#), um Daten (als JSON- oder Base64-kodierte Zeichenfolge) an den Zielkanal zu senden. Der Konnektor kann Rohdaten in ein Format

verarbeiten, das den API-Anforderungen entspricht. Der Konnektor puffert Eingabenachrichten in Warteschlangen pro Kanal und verarbeitet die Stapel asynchron. Sie bietet Parameter, mit denen Sie das Verhalten von Warteschlangen und Stapeln steuern und die Speicherbelegung beschränken können. Beispielsweise können Sie die maximale Warteschlangengröße, den Stapelintervall, die Größe des Arbeitsspeichers und die Anzahl der aktiven Kanäle konfigurieren.

Dieser Connector hat die folgenden Versionen.

Version	ARN
4	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTAnalytics/versions/4</code>
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTAnalytics/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTAnalytics/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/IoTAnalytics/versions/1</code>

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

 Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl als auch unterstützt [AWS IoT Analytics](#) werden.
- Alle zugehörigen AWS IoT Analytics Entitäten und Workflows werden erstellt und konfiguriert. Zu den Entitäten gehören Kanäle, Pipeline, Datastores und Datasets. Weitere Informationen finden Sie unter [AWS CLI](#) oder [Konsolenprozeduren](#) im AWS IoT Analytics -Benutzerhandbuch.

 Note

AWS IoT Analytics Zielkanäle müssen dasselbe Konto verwenden und sich im selben AWS-Region Connector befinden.

- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `iotanalytics:BatchPutMessage` Aktion auf Zielkanälen zulässt, wie im folgenden Beispiel für eine IAM-Richtlinie gezeigt. Die Kanäle müssen sich im aktuellen Bereich AWS-Konto und in der Region befinden.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Effect": "Allow",
      "Resource": [
```

```

        "arn:aws:iotanalytics:us-
east-1:123456789012:channel/channel_1_name",
        "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
    ]
}
]
}

```

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl als auch unterstützt [AWS IoT Analytics](#) werden.
- Alle zugehörigen AWS IoT Analytics Entitäten und Workflows werden erstellt und konfiguriert. Zu den Entitäten gehören Kanäle, Pipeline, Datastores und Datasets. Weitere Informationen finden Sie unter [AWS CLI](#) oder [Konsolenprozeduren](#) im AWS IoT Analytics -Benutzerhandbuch.

Note

AWS IoT Analytics Zielkanäle müssen dasselbe Konto verwenden und sich im selben AWS-Region Connector befinden.

- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `iotanalytics:BatchPutMessage` Aktion auf Zielkanälen zulässt, wie im folgenden Beispiel für eine IAM-Richtlinie gezeigt. Die Kanäle müssen sich im aktuellen Bereich AWS-Konto und in der Region befinden.

JSON

```

{
  "Version": "2012-10-17",

```

```

    "Statement": [
      {
        "Sid": "Stmt1528133056761",
        "Action": [
          "iotanalytics:BatchPutMessage"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:iotanalytics:us-
east-1:123456789012:channel/channel_1_name",
          "arn:aws:iotanalytics:us-east-1:123456789012:channel/channel_2_name"
        ]
      }
    ]
  }
}

```

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Parameters

MemorySize

Die Speichergröße (in KB), die diesem Konnektor zugewiesen werden soll.

Anzeigenname in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

PublishRegion

Das AWS-Region , in dem Ihre AWS IoT Analytics Kanäle erstellt wurden. Verwenden Sie dieselbe Region wie dem Konnektor.

 Note

Diese muss ebenfalls der Region für die Kanäle entsprechen, die in der [Gruppenrolle](#) angegeben sind.

Anzeigename in der AWS IoT Konsole: Region veröffentlichen

Erforderlich: false

Typ: string

Gültiges Muster: `^[a-z]{2}-[a-z]+-\d{1}`

PublishInterval

Das Intervall (in Sekunden), in dem ein Stapel empfangener Daten veröffentlicht wird AWS IoT Analytics.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall

Erforderlich: false

Typ: string

Standardwert: 1

Gültiges Muster: `^[0-9]+$`

IotAnalyticsMaxActiveChannels

Die maximale Anzahl von AWS IoT Analytics Kanälen, nach denen der Connector aktiv sucht. Diese muss größer als 0 und mindestens gleich der Anzahl der Kanäle sein, die voraussichtlich von dem Konnektor zu einem bestimmten Zeitpunkt veröffentlicht werden.

Sie können mit diesem Parameter den Speicherverbrauch beschränken, indem Sie die Gesamtzahl der Warteschlangen, die von dem Konnektor zu einem bestimmten Zeitpunkt verwaltet werden können, einschränken. Eine Warteschlange wird gelöscht, wenn alle Nachrichten darin gesendet wurden.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl aktiver Kanäle

Erforderlich: false

Typ: string

Standardwert: 50

Gültiges Muster: `^[0-9]*$`

`IotAnalyticsQueueDropBehavior`

Das Verhalten für das Löschen von Nachrichten aus einer Kanalwarteschlange, wenn die Warteschlange voll ist.

Anzeigename in der AWS IoT Konsole: Verhalten beim Löschen der Warteschlange

Erforderlich: false

Typ: string

Gültige Werte: `DROP_NEWEST` oder `DROP_OLDEST`.

Standardwert: `DROP_NEWEST`

Gültiges Muster: `^(DROP_NEWEST|DROP_OLDEST)$`

`IotAnalyticsQueueSizePerChannel`

Die maximale Anzahl der vor dem Senden oder Verwerfen der Nachrichten (pro Kanal) im Speicher gehaltenen Nachrichten. Dieser Wert muss größer als 0 sein.

Anzeigename in der AWS IoT Konsole: Maximale Warteschlangengröße pro Kanal

Erforderlich: false

Typ: string

Standardwert: 2048

Gültiges Muster: `^[0-9]*$`

`IotAnalyticsBatchSizePerChannel`

Die maximale Anzahl von Nachrichten, die in einer Batch-Anfrage an einen AWS IoT Analytics Kanal gesendet werden können. Dieser Wert muss größer als 0 sein.

Anzeigenname in der AWS IoT Konsole: Maximale Anzahl von Nachrichten, die pro Kanal gebündelt werden können

Erforderlich: false

Typ: string

Standardwert: 5

Gültiges Muster: `^$|^[1-9][0-9]*$`

`IotAnalyticsDefaultChannelName`

Der Name des AWS IoT Analytics Kanals, den dieser Connector für Nachrichten verwendet, die an ein vom Kunden definiertes Eingabethema gesendet werden.

Anzeigenname in der AWS IoT Konsole: Standardname des Kanals

Erforderlich: false

Typ: string

Gültiges Muster: `^[a-zA-Z0-9_]$`

`IsolationMode`

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigenname in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: false

Typ: string

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültiges Muster: `^NoContainer$|^GreengrassContainer$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den IoT Analytics Analytics-Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyIoTAnalyticsApplication",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTAnalytics/  
versions/3",  
      "Parameters": {  
        "MemorySize": "65535",  
        "PublishRegion": "us-west-1",  
        "PublishInterval": "2",  
        "IotAnalyticsMaxActiveChannels": "25",  
        "IotAnalyticsQueueDropBehavior": "DROP_OLDEST",  
        "IotAnalyticsQueueSizePerChannel": "1028",  
        "IotAnalyticsBatchSizePerChannel": "5",  
        "IotAnalyticsDefaultChannelName": "my_channel"  
      }  
    }  
  ]  
}'
```

Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Konnektor akzeptiert Daten auf vordefinierten und vom Kunden definierten MQTT-Themen. Publisher können Client-Geräte, Lambda-Funktionen oder andere Konnektoren sein.

Vordefinierte Themen

Der Konnektor unterstützt die folgenden beiden strukturierten MQTT-Themen, mit denen Herausgeber den Kanalnamen inline angeben können.

- Eine [formatierte Nachricht](#) zum `iotanalytics/channels/+/messages/put`-Thema. Die IoT-Daten in diesen Eingabenachrichten müssen als JSON- oder base64-kodierte Zeichenfolge formatiert sein.
- Eine unformatierte Nachricht auf dem `iotanalytics/channels/+/messages/binary/put`-Thema. Die zu diesem Thema erhalten Eingabenachrichten werden als binäre Daten behandelt können einen beliebigen Datentyp enthalten.

Wenn Sie vordefinierte Themen veröffentlichen möchten, ersetzen Sie den `+`-Platzhalter durch den Kanalnamen. Beispiel:

```
iotanalytics/channels/my_channel/messages/put
```

Kundendefinierte Themen

Der Konnektor unterstützt die `#`-Themasyntax, wodurch er Eingabenachrichten zu einem beliebigen MQTT-Thema annehmen kann, das Sie in einem Abonnement konfigurieren. Wir empfehlen, dass Sie einen Themenpfad angeben, anstatt nur den `#` Platzhalter in Ihren Abonnements zu verwenden. Diese Nachrichten werden an den Standardkanal gesendet, den Sie für den Connector angeben.

Eingabenachrichten zu den von Kunden definierten Themen werden als Binärdaten behandelt. Sie können eine beliebiges Nachrichtformat verwenden und beliebige Datentypen enthalten. Mit den vom Kunden definierten Themen können Sie Nachrichten von Geräten weiterleiten, die an feste Themen veröffentlicht werden. Sie können sie auch verwenden, um Eingabedaten von Client-Geräten zu akzeptieren, die die Daten nicht zu einer formatierten Nachricht verarbeiten können, um sie an den Connector zu senden.

Weitere Informationen zu Abonnements und MQTT-Themen finden Sie unter [the section called "Eingaben und Ausgaben"](#).

Die Gruppenrolle muss die `iotanalytics:BatchPutMessage`-Aktion auf allen Zielkanälen erlauben. Weitere Informationen finden Sie unter [the section called "Voraussetzungen"](#).

Themenfilter: `iotanalytics/channels/+/messages/put`

Mit diesem Thema können Sie formatierte Nachrichten an den Konnektor senden und dynamisch einen Zielkanal angeben. Dieses Thema ermöglicht Ihnen auch die Angabe einer ID, die in der Antwortausgabe zurückgegeben wird. Der Connector überprüft, IDs ob jede Nachricht in der ausgehenden BatchPutMessage Anfrage, an die er sendet, eindeutig ist. AWS IoT Analytics Eine Nachricht mit einer doppelten ID wird gelöscht.

Die Eingabedaten zu diesem Thema müssen das folgende Format aufweisen.

Nachrichten-Eigenschaften

`request`

Die an den angegebenen Kanal zu sendenden Daten.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`message`

Die Geräte- oder Sensordaten als JSON- oder base64-kodierte Zeichenfolge.

Erforderlich: `true`

Typ: `string`

`id`

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen. Wenn angegeben, wird die Eigenschaft `id` im Antwortobjekt auf diesen Wert gesetzt. Wenn Sie diese Eigenschaft auslassen, wird von dem Konnektor eine ID generiert.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `. *`

Beispieleingabe

```
{
  "request": {
    "message" : "{\"temp\":23.33}"
  }
}
```

```
  },  
  "id" : "req123"  
}
```

Themenfilter: `iotanalytics/channels/+/messages/binary/put`

Mit diesem Thema können Sie unformatierte Nachrichten an den Konnektor senden und dynamisch einen Zielkanal angeben.

Die Konnektor-Daten analysieren die zu diesem Thema empfangenen Eingabenachrichten nicht. Sie werden als Binärdaten behandelt. Vor dem Senden der Nachrichten an AWS IoT Analytics kodiert und formatiert der Connector sie entsprechend den BatchPutMessage API-Anforderungen:

- Der Konnektor führt eine base64-Verschlüsselung der Rohdaten durch und schließt diese verschlüsselte Nutzlast in einer ausgehenden BatchPutMessage-Anforderung ein.
- Der Konnektor generiert eine ID für jede Eingabenachricht und weist sie zu.



Note

In der Konnektor-Antwortausgabe ist keine ID-Korrelation für diese Eingabenachrichten beinhaltet.

Nachrichten-Eigenschaften

Keine.

Themenfilter: `#`

Verwenden Sie dieses Thema, um ein beliebiges Nachrichtenformat an den Standardkanal zu senden. Dies ist besonders nützlich, wenn Ihre Client-Geräte zu festen Themen veröffentlichen oder wenn Sie Daten von Client-Geräten, die die Daten nicht in das vom Connector [unterstützte Nachrichtenformat](#) verarbeiten können, an den Standardkanal senden möchten.

Sie definieren die Themensyntax in dem Abonnement, das Sie erstellen, um diesen Connector mit der Datenquelle zu verbinden. Wir empfehlen, dass Sie einen Themenpfad angeben, anstatt nur den # Platzhalter in Ihren Abonnements zu verwenden.

Die Konnektor-Daten analysieren die an dieses Eingabethema veröffentlichten Nachrichten nicht. Alle Eingabenachrichten werden als Binärdaten behandelt. Vor dem Senden der Nachrichten an AWS IoT Analytics kodiert und formatiert der Connector sie entsprechend den API-Anforderungen: BatchPutMessage

- Der Konnektor führt eine base64-Verschlüsselung der Rohdaten durch und schließt diese verschlüsselte Nutzlast in einer ausgehenden BatchPutMessage-Anforderung ein.
- Der Konnektor generiert eine ID für jede Eingabenachricht und weist sie zu.

 Note

In der Konnektor-Antwortausgabe ist keine ID-Korrelation für diese Eingabenachrichten beinhaltet.

Nachrichten-Eigenschaften

Keine.

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema. Diese Information enthält die Antwort, die von AWS IoT Analytics für jede Eingangsnachricht zurückgegeben wird, an die er empfängt und an die er sendet. AWS IoT Analytics

Themenfilter im Abonnement

`iotanalytics/messages/put/status`

Beispielausgabe: Erfolg

```
{
  "response" : {
    "status" : "success"
  },
  "id" : "req123"
}
```

Beispielausgabe: Fehler

```
{
  "response" : {
    "status" : "fail",
    "error" : "ResourceNotFoundException",
    "error_message" : "A resource with the specified name could not be found."
  },
  "id" : "req123"
}
```

```
}
```

Note

Wenn der Connector einen Fehler erkennt, der wiederholt werden kann (z. B. Verbindungsfehler), versucht er erneut, die Veröffentlichung im nächsten Batch durchzuführen. Der exponentielle Backoff wird vom SDK abgewickelt. AWS Anfragen mit wiederholbaren Fehlern werden zur weiteren Veröffentlichung entsprechend dem `IotAnalyticsQueueDropBehavior`-Parameter wieder der Kanalwarteschlange hinzugefügt.

Beispiel für eine Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-

Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.

- a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder `"Pinned": true` in der CLI).
- b. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
- c. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.

4. Stellen Sie die Gruppe bereit.

5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema `Ausgabedaten`, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder `"Pinned": false` in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'iotanalytics/channels/my_channel/messages/put'
```

```
def create_request_with_all_fields():
    return {
        "request": {
            "message" : "{\"temp\":23.33}"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Einschränkungen

Dieser Konnektor unterliegt den folgenden Einschränkungen.

- Alle Grenzwerte, die von der AWS SDK für Python (Boto3) für die AWS IoT Analytics [batch_put_message](#)Aktion festgelegt wurden.
- Alle von der AWS IoT Analytics [BatchPutMessage](#)API auferlegten Kontingente. Weitere Informationen finden Sie unter [Servicekontingente](#) für AWS IoT Analytics in der Allgemeine AWS-Referenz.
 - 100.000 Nachrichten pro Sekunde pro Kanal.
 - 100 Nachrichten pro Stapel.
 - 128 KB pro Nachricht.

Diese API verwendet Kanalnamen (keine Kanäle ARNs), sodass das Senden von Daten an regionsübergreifende oder kontoübergreifende Kanäle nicht unterstützt wird.

- Alle vom Core auferlegten Kontingente. AWS IoT Greengrass Weitere Informationen finden Sie unter [Service Quotas](#) für den AWS IoT Greengrass Kern in der Allgemeine AWS-Referenz.

Die folgenden Kontingente sind möglicherweise in besonderem Maße zutreffend:

- Die maximale Größe der von einem Gerät gesendeten Nachrichten ist 128 KB.

- Die maximale Größe der Nachrichtenwarteschlange im Greengrass Core-Router ist 2,5 MB.
- Die maximale Länge einer Thema-Zeichenfolge beträgt 256 Byte UTF-8-Zeichen.

Lizenzen

Der IoT Analytics Analytics-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
4	Fügt den <code>IsolationMode</code> Parameter hinzu, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.

Version	Änderungen
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Was ist AWS IoT Analytics?](#) im AWS IoT Analytics Benutzerhandbuch

IoT-Ethernet-IP-Protokolladapteranschluss

Der [IoT-Ethernet-IP-Protokolladapter-Anschluss](#) sammelt Daten von lokalen Geräten mithilfe des EtherNet/IP-Protokolls. Sie können diesen Connector verwenden, um Daten von mehreren Geräten zu sammeln und sie in einem StreamManager Nachrichtenstream zu veröffentlichen.

Sie können diesen Connector auch mit dem SiteWise IoT-Connector und Ihrem SiteWise IoT-Gateway verwenden. Ihr Gateway muss die Konfiguration für den Connector bereitstellen. Weitere Informationen finden [Sie unter Konfiguration einer EtherNet/IP \(EIP\) -Quelle](#) im SiteWise IoT-Benutzerhandbuch.

Note

Dieser Connector wird im Modus „[Keine Container-Isolierung](#)“ ausgeführt, sodass Sie ihn für eine AWS IoT Greengrass Gruppe bereitstellen können, die in einem Docker-Container ausgeführt wird.

Dieser Connector hat die folgenden Versionen.

Version	ARN
2 (empfohlen)	arn:aws:greengrass: <i>region</i> ::/connectors/IoTEIPProtocolAdaptor/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTEIPProtocolAdaptor/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 1 and 2

- AWS IoT Greengrass Kernsoftware v1.10.2 oder höher.
- Stream Manager ist für die AWS IoT Greengrass Gruppe aktiviert.
- Java 8 wurde auf dem Core-Gerät installiert und der PATH Umgebungsvariablen hinzugefügt.
- Mindestens 256 MB zusätzlicher RAM. Diese Anforderung gilt zusätzlich zu den Anforderungen an den AWS IoT Greengrass Kernspeicher.

Note

Dieser Connector ist nur in den folgenden Regionen verfügbar:

- cn-north-1
- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

Konnektor-Parameter

Dieser Konnektor unterstützt die folgenden Parameter:

LocalStoragePath

Das Verzeichnis auf dem AWS IoT Greengrass Host, in das der SiteWise IoT-Connector persistente Daten schreiben kann. Das Standardverzeichnis ist `/var/sitewise`.

Anzeigename in der AWS IoT Konsole: Lokaler Speicherpfad

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$|\/`.

ProtocolAdapterConfiguration

Der Satz von EtherNet/IP-Collector-Konfigurationen, von denen der Connector Daten sammelt oder mit denen er eine Verbindung herstellt. Dies kann eine leere Liste sein.

Anzeigename in der AWS IoT Konsole: Konfiguration des Protokolladapters

Erforderlich: `true`

Typ: Eine wohlgeformte JSON-Zeichenfolge, die den Satz der unterstützten Feedback-Konfigurationen definiert.

Das Folgende ist ein Beispiel für: `ProtocolAdapterConfiguration`

```
{
  "sources": [
    {
      "type": "EIPSource",
      "name": "TestSource",
      "endpoint": {
        "ipAddress": "52.89.2.42",
        "port": 44818
      },
      "destination": {
        "type": "StreamManager",
        "streamName": "MyOutput_Stream",

```

```

        "streamBufferSize": 10
    },
    "destinationPathPrefix": "EIPSource_Prefix",
    "propertyGroups": [
        {
            "name": "DriveTemperatures",
            "scanMode": {
                "type": "POLL",
                "rate": 10000
            },
            "tagPathDefinitions": [
                {
                    "type": "EIPTagPath",
                    "path": "arrayREAL[0]",
                    "dstDataType": "double"
                }
            ]
        }
    ]
}

```

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt einen Connector ConnectorDefinition mit einer ersten Version, die den IoT Ethernet IP Protocol Adapter Connector enthält.

```

aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
  "Connectors": [
    {
      "Id": "MyIoTEIPProtocolConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
      "Parameters": {
        "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
\"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
\"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
\"arrayREAL[0]\", \"dstDataType\": \"double\" }]]]]}",

```

```
        "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
    }
}
]
```

Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

Eingabedaten

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht Daten für `StreamManager`. Sie müssen den Zielnachrichtenstream konfigurieren. Die Ausgabenachrichten haben die folgende Struktur:

```
{
  "alias": "string",
  "messages": [
    {
      "name": "string",
      "value": boolean|double|integer|string,
      "timestamp": number,
      "quality": "string"
    }
  ]
}
```

Lizenzen

Der IoT Ethernet IP Protocol Adapter-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [EtherNet/IP-Client](#)
- [MapDB](#)
- [Elsa](#)

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen	Datum
2	Diese Version enthält Fehlerkorrekturen.	23. Dezember 2021
1	Erstversion.	15. Dezember 2020

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

SiteWise IoT-Anschluss

Der SiteWise IoT-Connector sendet lokale Geräte- und Gerätedaten an Anlagenimmobilien in AWS IoT SiteWise. Sie können diesen Connector verwenden, um Daten von mehreren OPC-UA-Servern zu sammeln und im IoT zu veröffentlichen. SiteWise Der Konnektor sendet die Daten an Objekteigenschaften in der aktuellen Region AWS-Konto und in der Region.

Note

IoT SiteWise ist ein vollständig verwalteter Service, der Daten von industriellen Geräten und Anlagen sammelt, verarbeitet und visualisiert. Sie können Komponenteneigenschaften konfigurieren, die Rohdaten verarbeiten, die von diesem Konnektor an die Messeigenschaften Ihrer Komponenten gesendet werden. Sie können beispielsweise eine Transformationseigenschaft definieren, die die Celsius-Temperaturdatenpunkte eines

Geräts in Fahrenheit konvertiert. Sie können auch eine Metrikeigenschaft definieren, die die durchschnittliche stündliche Temperatur berechnet. Weitere Informationen finden Sie unter [Was ist AWS IoT SiteWise?](#) im AWS IoT SiteWise -Benutzerhandbuch.

Der Connector sendet Daten SiteWise mit den OPC-UA-Datenstream-Pfaden, die von den OPC-UA-Servern gesendet werden, an das IoT. Beispielsweise könnte der Datenstream-Pfad `/company/windfarm/3/turbine/7/temperature` den Temperatursensor der Turbine #7 im Windpark #3 darstellen. Wenn der AWS IoT Greengrass Core die Verbindung zum Internet verliert, speichert der Connector Daten im Cache, bis er erfolgreich eine Verbindung zum herstellen kann. AWS Cloud Sie können die maximale Datenträgerpuffergröße konfigurieren, die zum Zwischenspeichern von Daten verwendet wird. Wenn die Cachegröße die maximale Datenträgerpuffergröße überschreitet, verwirft der Konnektor die ältesten Daten aus der Warteschlange.

Nachdem Sie den SiteWise IoT-Connector konfiguriert und bereitgestellt haben, können Sie ein Gateway und OPC-UA-Quellen in der [SiteWise IoT-Konsole](#) hinzufügen. Wenn Sie eine Quelle in der Konsole konfigurieren, können Sie die vom SiteWise IoT-Connector gesendeten OPC-UA-Datenstream-Pfade filtern oder mit einem Präfix versehen. Anweisungen zum Abschluss der Einrichtung des Gateways und der Quellen finden Sie unter [Hinzufügen des Gateways](#) im AWS IoT SiteWise -Benutzerhandbuch.

IoT SiteWise empfängt Daten nur aus Datenströmen, die Sie den Messeigenschaften von SiteWise IoT-Anlagen zugeordnet haben. Um Datenstreams Objekteigenschaften zuzuordnen, können Sie den Alias einer Eigenschaft so einstellen, dass er einem OPC-UA-Datenstream-Pfad entspricht. Weitere Informationen zum Definieren von Komponentenmodellen und zum Erstellen von Komponenten finden Sie unter [Modellieren von industriellen Komponenten](#) im AWS IoT SiteWise -Benutzerhandbuch.

Hinweise

Sie können den Stream Manager verwenden, um Daten SiteWise von anderen Quellen als OPC-UA-Servern in das IoT hochzuladen. Stream Manager bietet auch anpassbare Unterstützung für Persistenz- und Bandbreitenmanagement. Weitere Informationen finden Sie unter [Verwalten von Daten-Streams](#).

Dieser Connector wird im Modus [Keine Container-Isolierung](#) ausgeführt, sodass Sie ihn in einer Greengrass-Gruppe bereitstellen können, die in einem Docker-Container ausgeführt wird.

Dieser Connector hat die folgenden Versionen.

Version	ARN
12 (empfohlen)	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 12
11	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 11
10	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 10
9	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 9
8	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 8
7	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 7
6	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 6
5	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 5
4	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 4

Version	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 3
2	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 2
1	arn:aws:greengrass: <i>region</i> ::/connectors/IoTSiteWise/versions/ 1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 9, 10, 11, and 12

Important

Diese Version führt neue Anforderungen ein: AWS IoT Greengrass Kernsoftware v1.10.2 und [Stream Manager](#).

- AWS IoT Greengrass Kernsoftware v1.10.2.
- [Stream-Manager](#) muss für die Greengrass-Gruppe aktiviert sein.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl [IoT als auch IoT](#) unterstützt SiteWise werden.
- Der Greengrass-Gruppenrolle wurde eine IAM-Richtlinie hinzugefügt. Diese Rolle ermöglicht der AWS IoT Greengrass Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue` Aktion für das Ziel-Root-Asset und seine

untergeordneten Objekte, wie im folgenden Beispiel gezeigt. Sie können das Condition aus der Richtlinie entfernen, damit der Connector auf alle Ihre SiteWise IoT-Assets zugreifen kann.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Weitere Informationen finden Sie unter [Hinzufügen und Entfernen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Versions 6, 7, and 8

 Important

[Diese Version führt neue Anforderungen ein: AWS IoT Greengrass Kernsoftware v1.10.0 und Stream Manager.](#)

- AWS IoT Greengrass Kernsoftware v1.10.0.
- [Stream-Manager](#) muss für die Greengrass-Gruppe aktiviert sein.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl [IoT als auch IoT](#) unterstützt SiteWise werden.

- Der Greengrass-Gruppenrolle wurde eine IAM-Richtlinie hinzugefügt. Diese Rolle ermöglicht der AWS IoT Greengrass Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue` Aktion für das Ziel-Root-Asset und seine untergeordneten Objekte, wie im folgenden Beispiel gezeigt. Sie können das `Condition` aus der Richtlinie entfernen, damit der Connector auf alle Ihre SiteWise IoT-Assets zugreifen kann.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Weitere Informationen finden Sie unter [Hinzufügen und Entfernen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Version 5

- AWS IoT Greengrass Kernsoftware v1.9.4.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen `PATH` hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl [IoT als auch loT](#) unterstützt SiteWise werden.
- Der Greengrass-Gruppenrolle wurde eine IAM-Richtlinie hinzugefügt. Diese Rolle ermöglicht der AWS IoT Greengrass Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue` Aktion für das Ziel-Root-Asset und seine

untergeordneten Objekte, wie im folgenden Beispiel gezeigt. Sie können das `Condition` aus der Richtlinie entfernen, damit der Connector auf alle Ihre SiteWise IoT-Assets zugreifen kann.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Weitere Informationen finden Sie unter [Hinzufügen und Entfernen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Version 4

- AWS IoT Greengrass Kernsoftware v1.10.0.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen `PATH` hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl [IoT als auch IoT](#) unterstützt SiteWise werden.
- Der Greengrass-Gruppenrolle wurde eine IAM-Richtlinie hinzugefügt. Diese Rolle ermöglicht der AWS IoT Greengrass Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue` Aktion für das Ziel-Root-Asset und seine untergeordneten Objekte, wie im folgenden Beispiel gezeigt. Sie können das `Condition` aus der Richtlinie entfernen, damit der Connector auf alle Ihre SiteWise IoT-Assets zugreifen kann.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotsitewise:BatchPutAssetPropertyValue",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iotsitewise:assetHierarchyPath": [
            "/root node asset ID",
            "/root node asset ID/*"
          ]
        }
      }
    }
  ]
}
```

Weitere Informationen finden Sie unter [Hinzufügen und Entfernen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Version 3

- AWS IoT Greengrass Kernsoftware v1.9.4.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl [IoT als auch IoT](#) unterstützt SiteWise werden.
- Der Greengrass-Gruppenrolle wurde eine IAM-Richtlinie hinzugefügt. Diese Rolle ermöglicht der AWS IoT Greengrass Gruppe den Zugriff auf die `iotsitewise:BatchPutAssetPropertyValue` Aktion für das Ziel-Root-Asset und seine untergeordneten Objekte, wie im folgenden Beispiel gezeigt. Sie können das Condition aus der Richtlinie entfernen, damit der Connector auf alle Ihre SiteWise IoT-Assets zugreifen kann.

JSON

```
{
```



```

"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": "iotsitewise:BatchPutAssetPropertyValue",
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "iotsitewise:assetHierarchyPath": [
          "/root node asset ID",
          "/root node asset ID/*"
        ]
      }
    }
  }
]
}

```

Weitere Informationen finden Sie unter [Hinzufügen und Entfernen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Versions 1 and 2

- AWS IoT Greengrass Kernsoftware v1.9.4.
- Java 8 auf dem Core-Gerät installiert und der Umgebungsvariablen PATH hinzugefügt.
- Dieser Connector kann nur in Amazon Web Services Services-Regionen verwendet werden, in denen [AWS IoT Greengrass](#) sowohl [IoT als auch IoT](#) unterstützt SiteWise werden.
- Der Greengrass-Gruppenrolle wurde eine IAM-Richtlinie hinzugefügt, die den Zugriff auf das Ziel-Root-Asset AWS IoT Core und seine untergeordneten Objekte und die entsprechenden `iotsitewise:BatchPutAssetPropertyValue` Aktionen ermöglicht, wie im folgenden Beispiel gezeigt. Sie können das Condition aus der Richtlinie entfernen, damit der Connector auf alle Ihre SiteWise IoT-Assets zugreifen kann.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",

```

```

        "Action": "iotsitewise:BatchPutAssetPropertyValue",
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "iotsitewise:assetHierarchyPath": [
                    "/root node asset ID",
                    "/root node asset ID/*"
                ]
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:DescribeEndpoint",
                "iot:Publish",
                "iot:Receive",
                "iot:Subscribe"
            ],
            "Resource": "*"
        }
    ]
}

```

Informationen finden Sie im Abschnitt [Hinzufügen und Entfernen von IAM-Identitätsberechtigungen](#) im -IAM-Benutzerhandbuch.

Parameters

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

SiteWiseLocalStoragePath

Das Verzeichnis auf dem AWS IoT Greengrass Host, in das der SiteWise IoT-Connector persistente Daten schreiben kann. Standardeinstellung: `/var/sitewise`.

Anzeigename in der AWS IoT Konsole: Lokaler Speicherpfad

Erforderlich: false

Typ: string

Gültiges Muster: `^\s*$|\/.`

`AWSecretsArnList`

Eine Liste von Geheimnissen AWS Secrets Manager, die jeweils ein Schlüssel-Wert-Paar aus OPC-UA-Benutzername und Passwort enthalten. Alle Secrets müssen vom Typ Schlüssel-Wert-Paar sein.

Anzeigename in der AWS IoT Konsole: Liste von vier OPC-UA-Geheimnissen ARNs
username/password

Erforderlich: `false`

Typ: `JsonArrayOfStrings`

Gültiges Muster: `\[( ? , ? ? \"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\\\]+\\/*)[a-zA-Z0-9\\/_+=, .@\\-]+-[a-zA-Z0-9]+)*\\\" )*\]`

`MaximumBufferSize`

Die maximale Größe in GB für die Nutzung von SiteWise IoT-Festplatten. Standardmäßig 10 GB.

Anzeigename in der AWS IoT Konsole: Maximale Festplattenpuffergröße

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$| [0-9]+`

Version 1

`SiteWiseLocalStoragePath`

Das Verzeichnis auf dem AWS IoT Greengrass Host, in das der SiteWise IoT-Connector persistente Daten schreiben kann. Standardeinstellung: `/var/sitewise`.

Anzeigename in der AWS IoT Konsole: Lokaler Speicherpfad

Erforderlich: `false`

Typ: string

Gültiges Muster: `^\s*$|\/.`

SiteWiseOpcuaUserIdentityTokenSecretArn

Das darin enthaltene Geheimnis enthält AWS Secrets Manager das Schlüssel-Wert-Paar aus OPC-UA-Benutzername und Passwort. Dieses Secret muss ein Schlüssel-Wert-Paar-Typ-Secret sein.

Anzeigename in der AWS IoT Konsole: ARN von username/password OPC-UA Secret

Erforderlich: false

Typ: string

Gültiges Muster: `^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\-\\+\\/]*[a-zA-Z0-9/_+=, .@\\-]+-[a-zA-Z0-9]+`

SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId

Die geheime Ressource in der AWS IoT Greengrass Gruppe, die auf einen OPC-UA-Benutzernamen und ein geheimes Passwort verweist.

Anzeigename in der AWS IoT Konsole: Geheime username/password OPC-UA-Ressource

Erforderlich: false

Typ: string

Gültiges Muster: `^$|.+`

MaximumBufferSize

Die maximale Größe in GB für die Nutzung von SiteWise IoT-Festplatten. Standardmäßig 10 GB.

Anzeigename in der AWS IoT Konsole: Maximale Festplattenpuffergröße

Erforderlich: false

Typ: string

Gültiges Muster: `^\s*$| [0-9]+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende AWS CLI Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den SiteWise IoT-Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyIoTSiteWiseConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/  
versions/11"  
    }  
  ]  
}'
```

Note

Die Lambda-Funktionen in diesem Konnektor haben einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht keine MQTT-Nachrichten als Ausgabedaten.

Einschränkungen

Dieser Anschluss unterliegt den folgenden allen vom IoT auferlegten Grenzwerten SiteWise, einschließlich der folgenden. Weitere Informationen finden Sie unter [AWS IoT SiteWise Endpunkte und Kontingente](#) im. Allgemeine AWS-Referenz

- Maximale Anzahl von Gateways pro. AWS-Konto
- Maximale Anzahl von OPC-UA-Quellen pro Gateway.
- Maximale Rate an gespeicherten timestamp-quality-value (TQV) Datenpunkten pro. AWS-Konto
- Maximale Rate der pro Objekteigenschaft gespeicherten TQV-Datenpunkte.

Lizenzen

Version 9, 10, 11, and 12

Der SiteWise IoT-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [MapDB](#)
- [Elsa](#)
- [Verfinstere Milo](#)

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Versions 6, 7, and 8

Der SiteWise IoT-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [Milo](#) / EDL 1.0

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Versions 1, 2, 3, 4, and 5

Der SiteWise IoT-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [Milo](#) / EDL 1.0
- [Chronicle-Queue](#)/Apache-Lizenz 2.0

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen	Date
12	• Diese Version enthält Fehlerkorrekturen.	22. Dezember 2021
11	• Support für Zeichenketten, die versteckte oder nicht druckbare Zeichen enthalten. Versteckte und nicht druckbare Zeichen werden automatisch entfernt, bevor die Zeichenketten an die gesendet werden. AWS Cloud • Es wurde ein Problem behoben, das dazu führte, dass das SiteWise IoT-Gateway ungültige Anfragen immer wieder wiederholte. • Es wurde ein Problem behoben, das zu einem beschädigten Checkpoint führte, wenn das SiteWise IoT-Gateway mit einer Hochfrequenzdatenquelle verbunden war. • Die Fehlermeldungen wurden verbessert, um bei der Fehlerbehebung bei der Gateway-Konfiguration zu helfen.	24. März 2021

Version	Änderungen	Date
10	KonfiguriertStreamManager, um die Handhabung zu verbessern, wenn die Quellverbindung verloren geht und wieder hergestellt wird. Diese Version akzeptiert auch OPC-UA-Werte mit einem ServerTimestamp wenn kein SourceTimestamp verfügbar ist.	22. Januar 2021
9	Support für benutzerdefinierte StreamManager Greengrass-Stream-Ziele, OPC-UA-Deadbanding, benutzerdefinierten Scanmodus und benutzerdefinierte Scanrate eingeführt. Beinhaltet auch eine verbesserte Leistung bei Konfigurationsupdates, die über das SiteWise IoT-Gateway vorgenommen werden.	15. Dezember 2020
8	Verbesserte Stabilität bei intermittierender Netzwerkkonnektivität des Connectors.	19. November 2020
7	Ein Problem mit Gateway-Metriken wurde behoben.	14. August 2020

Version	Änderungen	Date
6	Unterstützung für CloudWatch Metriken und automatische Erkennung neuer OPC-UA-Tags hinzugefügt. Diese Version erfordert Stream Manager und AWS IoT Greengrass Core-Software v1.10.0 oder höher.	29. April 2020
5	Ein Kompatibilitätsproblem mit der AWS IoT Greengrass Core-Software v1.9.4 wurde behoben.	12. Februar 2020
4	Ein Problem mit der erneuten Verbindung des OPC-UA Servers wurde behoben.	7. Februar 2020
3	Die <code>iot:*</code> -Berechtigungsanforderung wurde entfernt.	17. Dezember 2019
2	Unterstützung für mehrere OPC-UA-Secret-Ressourcen hinzugefügt.	10. Dezember 2019
1	Erstversion.	02. Dezember 2019

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

- Weitere Informationen finden Sie in den folgenden Themen im AWS IoT SiteWise - Benutzerhandbuch:
 - [Was ist AWS IoT SiteWise?](#)
 - [Verwenden eines Gateways](#)
 - [CloudWatch Gateway-Metriken](#)
 - [Fehlerbehebung bei einem SiteWise IoT-Gateway](#)

Kinesis Firehose

Der Kinesis [Firehose-Konnektor](#) veröffentlicht Daten über einen Amazon Firehose Firehose-Lieferstream an Ziele wie Amazon S3, Amazon Redshift oder Amazon Service. OpenSearch

Dieser Connector ist ein Datenproduzent für einen Kinesis-Lieferstream. Er empfängt Eingabedaten zu einem MQTT-Thema und sendet die Daten an einen angegebenen Bereitstellungs-Stream. Der Bereitstellungs-Stream sendet dann den Datensatz an das konfigurierte Ziel (z. B. einen S3-Bucket).

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/2

Version	ARN
1	arn:aws:greengrass: <i>region</i> ::/connectors/KinesisFirehose/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 4 - 5

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Ein konfigurierter Kinesis-Lieferstream. Weitere Informationen finden Sie unter [Creating a Amazon Data Firehose Delivery Stream](#) im Amazon Kinesis Firehose Developer Guide.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `firehose:PutRecordBatch` Aktionen `firehose:PutRecord` und für den Zielzustellungsstream zulässt, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

JSON

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "Stmt1528133056761",
        "Action": [
          "firehose:PutRecord",
          "firehose:PutRecordBatch"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-
name"
        ]
      }
    ]
  }
}

```

Mit diesem Konnektor können Sie den standardmäßigen Bereitstellungs-Stream in der Nutzlast der Input-Message dynamisch überschreiben. Wenn Ihre Implementierung diese Funktion verwendet, sollte die IAM-Richtlinie alle Zielstreams als Ressourcen enthalten. Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Versions 2 - 3

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Ein konfigurierter Kinesis-Lieferstream. Weitere Informationen finden Sie unter [Creating a Amazon Data Firehose Delivery Stream](#) im Amazon Kinesis Firehose Developer Guide.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `firehose:PutRecordBatch` Aktionen `firehose:PutRecord` und für den Zielzustellungsstream zulässt, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-  
name"
      ]
    }
  ]
}
```

Mit diesem Konnektor können Sie den standardmäßigen Bereitstellungs-Stream in der Nutzlast der Input-Message dynamisch überschreiben. Wenn Ihre Implementierung diese Funktion verwendet, sollte die IAM-Richtlinie alle Zielstreams als Ressourcen enthalten. Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Version 1

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Ein konfigurierter Kinesis-Lieferstream. Weitere Informationen finden Sie unter [Creating a Amazon Data Firehose Delivery Stream](#) im Amazon Kinesis Firehose Developer Guide.

- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `firehose:PutRecord` Aktion auf dem Zielzustellungsstream zulässt, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1528133056761",
      "Action": [
        "firehose:PutRecord"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-  
name"
      ]
    }
  ]
}
```

Mit diesem Konnektor können Sie den standardmäßigen Bereitstellungs-Stream in der Nutzlast der Input-Message dynamisch überschreiben. Wenn Ihre Implementierung diese Funktion verwendet, sollte die IAM-Richtlinie alle Zielstreams als Ressourcen enthalten. Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Versions 5

DefaultDeliveryStreamArn

Der ARN des Standard-Firehose-Lieferstreams, an den Daten gesendet werden sollen. Der Ziel-Stream kann durch die Eigenschaft `delivery_stream_arn` in der Nutzlast der Eingangsnachricht überschrieben werden.

Note

Die Gruppenrolle muss die entsprechenden Aktionen für alle Zielbereitstellungsströme ermöglichen. Weitere Informationen finden Sie unter [the section called "Voraussetzungen"](#).

Anzeigename in der AWS IoT Konsole: Standard-Lieferstream-ARN

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `arn:aws:firehose:([a-z]{2}-[a-z]+\-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

DeliveryStreamQueueSize

Die maximale Anzahl der im Speicher behaltenen Datensätze, bevor neue Datensätze für den gleichen Bereitstellungsstrom abgelehnt werden. Der minimale Wert beträgt 2000.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl der zu puffernden Datensätze (pro Stream)

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

MemorySize

Die Speichergröße (in KB), die diesem Konnektor zugewiesen werden soll.

Anzeigename in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

PublishInterval

Das Intervall (in Sekunden) für die Veröffentlichung von Datensätzen in Firehose. Setzen Sie diesen Wert auf 0, um die Stapelverarbeitung zu deaktivieren.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall

Erforderlich: true

Typ: string

Zulässige Werte: 0 - 900

Gültiges Muster: `[0-9] | [1-9]\\d | [1-9]\\d\\d | 900`

IsolationMode

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: false

Typ: string

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültiges Muster: `^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

DefaultDeliveryStreamArn

Der ARN des Standard-Firehose-Lieferstreams, an den Daten gesendet werden sollen. Der Ziel-Stream kann durch die Eigenschaft `delivery_stream_arn` in der Nutzlast der Eingangsnachricht überschrieben werden.

 Note

Die Gruppenrolle muss die entsprechenden Aktionen für alle Zielbereitstellungsströme ermöglichen. Weitere Informationen finden Sie unter [the section called "Voraussetzungen"](#).

Anzeigename in der AWS IoT Konsole: Standard-Lieferstream-ARN

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-\.]+)$`

DeliveryStreamQueueSize

Die maximale Anzahl der im Speicher behaltenen Datensätze, bevor neue Datensätze für den gleichen Bereitstellungsstrom abgelehnt werden. Der minimale Wert beträgt 2000.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl der zu puffernden Datensätze (pro Stream)

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

MemorySize

Die Speichergröße (in KB), die diesem Konnektor zugewiesen werden soll.

Anzeigename in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

PublishInterval

Das Intervall (in Sekunden) für die Veröffentlichung von Datensätzen in Firehose. Setzen Sie diesen Wert auf 0, um die Stapelverarbeitung zu deaktivieren.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall

Erforderlich: true

Typ: string

Zulässige Werte: 0 - 900

Gültiges Muster: `[0-9] | [1-9]\\d | [1-9]\\d\\d | 900`

Version 1

DefaultDeliveryStreamArn

Der ARN des Standard-Firehose-Lieferstreams, an den Daten gesendet werden sollen. Der Ziel-Stream kann durch die Eigenschaft `delivery_stream_arn` in der Nutzlast der Eingangsnachricht überschrieben werden.

Note

Die Gruppenrolle muss die entsprechenden Aktionen für alle Zielbereitstellungsströme ermöglichen. Weitere Informationen finden Sie unter [the section called "Voraussetzungen"](#).

Anzeigename in der AWS IoT Konsole: Standard-Lieferstream-ARN

Erforderlich: true

Typ: string

Gültiges Muster: `arn:aws:firehose:([a-z]{2}-[a-z]+\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

Beispiel für Connector erstellen (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyKinesisFirehoseConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
      "Parameters": {
        "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
        "DeliveryStreamQueueSize": "5000",
        "MemorySize": "65535",
        "PublishInterval": "10",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}'
```

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Konnektor akzeptiert Stream-Inhalte zu MQTT-Themen und sendet die Inhalte dann an den Ziel-Bereitstellungs-Stream. Er akzeptiert zwei Arten von Eingabedaten:

- JSON-Daten zum `kinesisfirehose/message`-Thema.

- Binäre Daten zum `kinesisfirehose/message/binary/#`-Thema.

Versions 2 - 5

Themenfilter: `kinesisfirehose/message`

Verwenden Sie dieses Thema, um eine Nachricht zu senden, die JSON-Daten enthält.

Nachrichten-Eigenschaften

`request`

Die zu sendenden Daten an den Lieferstrom und den Ziel-Lieferstrom, falls sie sich vom Standard-Stream unterscheiden.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`data`

Die Daten, die an den Lieferstrom gesendet werden sollen.

Erforderlich: `true`

Typ: `string`

`delivery_stream_arn`

Der ARN des Kinesis-Ziel-Lieferstreams. Fügen Sie diese Eigenschaft hinzu, um den standardmäßigen Lieferstrom zu überschreiben.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-\.]+)$`

`id`

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen. Wenn angegeben, wird die Eigenschaft `id` im Antwortobjekt auf diesen Wert gesetzt. Wenn Sie diese Funktion

nicht verwenden, können Sie diese Eigenschaft weglassen oder eine leere Zeichenkette angeben.

Erforderlich: false

Typ: string

Gültiges Muster: .*

Beispieleingabe

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Themenfilter: `kinesisfirehose/message/binary/#`

Verwenden Sie dieses Thema, um eine Nachricht zu senden, die binäre Daten enthält. Der Konnektor analysiert keine binären Daten. Die Daten werden unverändert gestreamt.

Um die Eingabeaufforderung einer Ausgabeaufgabe zuzuordnen, ersetzen Sie den Platzhalter # im Meldungsthema durch eine beliebige Anforderungs-ID. Wenn Sie beispielsweise eine Nachricht an `kinesisfirehose/message/binary/request123` veröffentlichen, wird die Eigenschaft `id` im Antwortobjekt auf `request123` gesetzt.

Wenn Sie eine Anfrage nicht auf eine Antwort abbilden möchten, können Sie Ihre Nachrichten unter `kinesisfirehose/message/binary/` veröffentlichen. Achten Sie darauf, dass Sie den nachlaufenden Schrägstrich verwenden.

Version 1

Themenfilter: `kinesisfirehose/message`

Verwenden Sie dieses Thema, um eine Nachricht zu senden, die JSON-Daten enthält.

Nachrichten-Eigenschaften

`request`

Die zu sendenden Daten an den Lieferstrom und den Ziel-Lieferstrom, falls sie sich vom Standard-Stream unterscheiden.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`data`

Die Daten, die an den Lieferstrom gesendet werden sollen.

Erforderlich: `true`

Typ: `string`

`delivery_stream_arn`

Der ARN des Kinesis-Ziel-Lieferstreams. Fügen Sie diese Eigenschaft hinzu, um den standardmäßigen Lieferstrom zu überschreiben.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-\.]+)$`

`id`

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen. Wenn angegeben, wird die Eigenschaft `id` im Antwortobjekt auf diesen Wert gesetzt. Wenn Sie diese Funktion nicht verwenden, können Sie diese Eigenschaft weglassen oder eine leere Zeichenkette angeben.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `.*`

Beispieleingabe

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-
id:deliverystream/stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Themenfilter: `kinesisfirehose/message/binary/#`

Verwenden Sie dieses Thema, um eine Nachricht zu senden, die binäre Daten enthält. Der Konnektor analysiert keine binären Daten. Die Daten werden unverändert gestreamt.

Um die Eingabeaufforderung einer Ausgabeaufgabe zuzuordnen, ersetzen Sie den Platzhalter # im Meldungsthema durch eine beliebige Anforderungs-ID. Wenn Sie beispielsweise eine Nachricht an `kinesisfirehose/message/binary/request123` veröffentlichen, wird die Eigenschaft `id` im Antwortobjekt auf `request123` gesetzt.

Wenn Sie eine Anfrage nicht auf eine Antwort abbilden möchten, können Sie Ihre Nachrichten unter `kinesisfirehose/message/binary/` veröffentlichen. Achten Sie darauf, dass Sie den nachlaufenden Schrägstrich verwenden.

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema.

Versions 2 - 5

Themenfilter im Abonnement

```
kinesisfirehose/message/status
```

Beispielausgabe

Die Antwort enthält den Status jedes im Stapel gesendeten Datensatzes.

```
{
```

```
"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

Wenn der Connector einen Fehler erkennt, der wiederholt werden kann (z. B. Verbindungsfehler), versucht er die Veröffentlichung im nächsten Batch erneut. Der exponentielle Backoff wird vom SDK abgewickelt. AWS Anfragen, die mit wiederholbaren Fehlern fehlschlagen, werden zur weiteren Veröffentlichung wieder am Ende der Warteschlange hinzugefügt.

Version 1

Themenfilter im Abonnement

kinesisfirehose/message/status

Beispielausgabe: Erfolg

```
{  
  "response": {  
    "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",  
    "status": "success"  
  },  
}
```



```
{  
  "id": "request123"  
}
```

Beispielausgabe: Fehler

```
{  
  "response" : {  
    "error": "ResourceNotFoundException",  
    "error_message": "An error occurred (ResourceNotFoundException) when  
calling the PutRecord operation: Firehose test1 not found under account  
123456789012.",  
    "status": "fail"  
  },  
  "id": "request123"  
}
```

Beispiel für eine Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder `"Pinned": true` in der CLI).
 - b. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - c. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [JSON-Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder `"Pinned": false` in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor. Diese Nachricht enthält JSON-Daten.

```
import greengrasssdk
import time
import json
```

```
iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der Kinesis Firehose-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
5	Der Parameter <code>IsolationMode</code> wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
4	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
3	Korrektur, um übermäßige Protokollierung und andere kleinere Fehlerbehebungen zu reduzieren.
2	Unterstützung für das Senden von Batch-Datensätzen an Firehose in einem bestimmten Intervall hinzugefügt. • Außerdem ist die <code>firehose:PutRecordBatch</code> -Aktion in der Gruppenrolle erforderlich. • Neue <code>MemorySize</code> -, <code>DeliveryStreamQueueSize</code> - und <code>PublishInterval</code> -Parameter. • Die Ausgabenachricht enthält ein Array von Statusantworten für die veröffentlichten Datensätze.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Was ist Amazon Kinesis Data Firehose?](#) im Amazon Kinesis Developer Guide

ML Feedback-Anschluss

Warning

Dieser Konnektor befindet sich in der Phase der verlängerten Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der ML-Feedback-Konnektor erleichtert den Zugriff auf Ihre Modelldaten für maschinelles Lernen (ML) für das erneute Trainieren und Analysieren von Modellen. Der Konnektor:

- Lädt Eingabedaten (Beispiele), die von Ihrem ML-Modell verwendet werden, auf Amazon S3 hoch. Die Modelleingabe kann in einem beliebigen Format vorliegen, z. B. in Form von Bildern, im JSON-Format oder als Audio. Nachdem Beispiele in die Cloud hochgeladen wurden, können Sie diese verwenden, um das Modell neu zu schulen und so die Genauigkeit und Präzision seiner Prognosen zu verbessern. Sie können beispielsweise [SageMaker KI Ground Truth](#) verwenden, um Ihre Proben zu kennzeichnen, und [SageMaker KI](#), um das Modell neu zu trainieren.
- Veröffentlicht die Prognoseergebnisse aus dem Modell als MQTT-Nachrichten. Auf diese Weise können Sie die Inferenzqualität Ihres Modells in Echtzeit überwachen und analysieren. Sie können die Prognoseergebnisse auch speichern und mit ihrer Hilfe Trends im Laufe der Zeit analysieren.
- Veröffentlicht Metriken zu Beispiel-Uploads und Beispieldaten auf Amazon CloudWatch.

Um diesen Konnektor zu konfigurieren, beschreiben Sie Ihre unterstützten Feedback-Konfigurationen im JSON-Format. Eine Feedback-Konfiguration definiert Eigenschaften wie den Amazon S3-Ziel-Bucket, den Inhaltstyp und die [Sampling-Strategie](#). (Mit einer Samplingstrategie wird ermittelt, welche Beispiele hochgeladen werden sollen.)

Sie können den ML-Feedback-Connector in den folgenden Szenarien verwenden:

- Mit benutzerdefinierten Lambda-Funktionen. Ihre lokalen Inferenz-Lambda-Funktionen verwenden das AWS IoT Greengrass Machine Learning SDK, um diesen Konnektor aufzurufen und die Ziel-Feedback-Konfiguration, die Modelleingabe und die Modellausgabe (Vorhersageergebnisse) weiterzuleiten. Ein Beispiel finden Sie unter [the section called “Beispiel für die Verwendung”](#).
- Mit dem [ML Image Classification Connector \(v2\)](#). Um diesen Konnektor mit dem ML Image Classification-Konnektor zu verwenden, konfigurieren Sie den `MLFeedbackConnectorConfigId` Parameter für den ML Image Classification-Konnektor.
- Mit dem [ML Object Detection-Konnektor](#). Um diesen Konnektor mit dem ML-Objekterkennungs-Konnektor zu verwenden, konfigurieren Sie den `MLFeedbackConnectorConfigId` Parameter für den ML-Objekterkennungs-Konnektor.

ARN: `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen `PATH` hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Ein oder mehrere Amazon S3 S3-Buckets. Wie viele Buckets Sie verwenden, hängt von Ihrer Samplingstrategie ab.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `s3:PutObject` Aktion für Objekte im Amazon S3 S3-Ziel-Bucket zulässt, wie in der folgenden IAM-Beispielrichtlinie dargestellt.

JSON

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

Die Richtlinie sollte alle Ziel-Buckets als Ressourcen enthalten. Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

- Der [CloudWatch Metrics-Konnektor](#) wurde der Greengrass-Gruppe hinzugefügt und konfiguriert. Nur erforderlich, wenn Sie die Berichterstellungsfunktion für Metriken verwenden möchten.
- AWS IoT Greengrass Für die Interaktion mit diesem Connector ist das [Machine Learning SDK](#) v1.1.0 erforderlich.

Parameters

FeedbackConfigurationMap

Eine Reihe von einer oder mehreren Feedback-Konfigurationen, die der Connector verwenden kann, um Proben auf Amazon S3 hochzuladen. Eine Feedback-Konfiguration definiert Parameter wie z. B. den Ziel-Bucket, den Inhaltstyp und die [Samplingstrategie](#). Wenn dieser Konnektor aufgerufen wird, spezifiziert die aufrufende Lambda-Funktion oder der aufrufende Connector eine Ziel-Feedback-Konfiguration.

Anzeigenname in der AWS IoT Konsole: Feedback-Konfigurationsübersicht

Erforderlich: `true`

Typ: Eine wohlgeformte JSON-Zeichenfolge, die den Satz der unterstützten Feedback-Konfigurationen definiert. Ein Beispiel finden Sie unter [the section called "FeedbackConfigurationMap Beispiel"](#).

Für die ID eines Feedback-Konfigurationsobjekts gelten folgende Anforderungen.

Die ID:

- Muss für alle Konfigurationsobjekte eindeutig sein.
- Muss mit einem Buchstaben oder einer Zahl beginnen. Kann Kleinbuchstaben und Großbuchstaben, Zahlen und Bindestriche enthalten.
- Muss zwischen 2 und 63 Zeichen lang sein.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Beispiele: `MyConfig0`, `config-a`, `12id`.

Der Text eines Feedback-Konfigurationsobjekts enthält die folgenden Eigenschaften.

`s3-bucket-name`

Der Name des Amazon S3-Ziel-Buckets.

 Note

Die Gruppenrolle muss die Aktion `s3:PutObject` auf allen Ziel-Buckets erlauben. Weitere Informationen finden Sie unter [the section called “Voraussetzungen”](#).

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^[a-z0-9\.\-]{3,63}$`

`content-type`

Der Inhaltstyp der hochzuladenden Beispiele. Alle Inhalte für eine einzelne Feedback-Konfiguration müssen denselben Typ aufweisen.

Erforderlich: `true`

Typ: `string`

Beispiele: `image/jpeg`, `application/json`, `audio/ogg`.

`s3-prefix`

Das Schlüsselpräfix, das für hochgeladene Beispiele verwendet werden soll. Ein Präfix ist einem Verzeichnisnamen ähnlich. Es bietet Ihnen die Möglichkeit, ähnliche Daten unter demselben Verzeichnis in einem Bucket zu speichern. Weitere Informationen finden Sie unter [Objektschlüssel und Metadaten](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Erforderlich: `false`

Typ: `string`

`file-ext`

Die Dateierweiterung, die für hochgeladene Beispiele verwendet werden soll. Es muss sich um eine gültige Dateierweiterung für den Inhaltstyp handeln.

Erforderlich: `false`

Typ: `string`

Beispiele: jpg, json, ogg.

sampling-strategy

Die [Samplingstrategie](#) zum Filtern der hochzuladenden Beispiele. Wenn nicht angegeben, versucht der Konnektor, alle empfangenen Beispiele hochzuladen.

Erforderlich: false

Typ: Eine wohlgeformte JSON-Zeichenfolge, die die folgenden Eigenschaften enthält.

strategy-name

Der Name der Samplingstrategie.

Erforderlich: true

Typ: string

Gültige Werte: RANDOM_SAMPLING, LEAST_CONFIDENCE, MARGIN oder ENTROPY

rate

Die Rate für die [Random](#)-Samplingstrategie.

Erforderlich: true wenn strategy-name jaRANDOM_SAMPLING.

Typ: number

Zulässige Werte: 0.0 - 1.0

threshold

Der Schwellenwert für die Samplingstrategie [Least Confidence \(Geringste Zuverlässigkeit\)](#), [Margin \(Abstand\)](#) oder [Entropy \(Entropie\)](#).

Erforderlich: true wenn strategy-name ist LEAST_CONFIDENCEMARGIN, oderENTROPY.

Typ: number

Zulässige Werte:

- 0.0 - 1.0 für die Strategie LEAST_CONFIDENCE oder MARGIN.

- 0.0 - no limit für die Strategie ENTROPY.

RequestLimit

Die maximale Anzahl von Anforderungen, die der Konnektor gleichzeitig verarbeiten kann.

Mithilfe dieses Parameters können Sie die Speicherbelegung einschränken, indem Sie die Anzahl der Anforderungen begrenzen, die der Konnektor gleichzeitig verarbeitet. Über dieses Limit hinausgehende Anforderungen werden ignoriert.

Anzeigename in der AWS IoT Konsole: Limit anfordern

Erforderlich: false

Typ: string

Zulässige Werte: 0 - 999

Gültiges Muster: ^\$|^[0-9]{1,3}\$

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine ConnectorDefinition mit einer ersten Version, die den ML-Feedback-Konnektor enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyMLFeedbackConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
      "Parameters": {
        "FeedbackConfigurationMap": "{ \"RandomSamplingConfiguration\":
{ \"s3-bucket-name\": \"my-aws-bucket-random-sampling\", \"content-type\":
\"image/png\", \"file-ext\": \"png\", \"sampling-strategy\": { \"strategy-name
\": \"RANDOM_SAMPLING\", \"rate\": 0.5 } }, \"LeastConfidenceConfiguration\": {
  \"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\", \"content-type\":
\"image/png\", \"file-ext\": \"png\", \"sampling-strategy\": { \"strategy-name\":
\"LEAST_CONFIDENCE\", \"threshold\": 0.4 } } }",
        "RequestLimit": "10"
      }
    }
  ]
}
```

```
    }  
  ]  
}'
```

FeedbackConfigurationMap Beispiel

Im Folgenden sehen Sie einen erweiterten Beispielwert für den Parameter `FeedbackConfigurationMap`. Dieses Beispiel enthält mehrere Feedback-Konfigurationen, die verschiedene Samplingstrategien verwenden.

```
{  
  "ConfigID1": {  
    "s3-bucket-name": "my-aws-bucket-random-sampling",  
    "content-type": "image/png",  
    "file-ext": "png",  
    "sampling-strategy": {  
      "strategy-name": "RANDOM_SAMPLING",  
      "rate": 0.5  
    }  
  },  
  "ConfigID2": {  
    "s3-bucket-name": "my-aws-bucket-margin-sampling",  
    "content-type": "image/png",  
    "file-ext": "png",  
    "sampling-strategy": {  
      "strategy-name": "MARGIN",  
      "threshold": 0.4  
    }  
  },  
  "ConfigID3": {  
    "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",  
    "content-type": "image/png",  
    "file-ext": "png",  
    "sampling-strategy": {  
      "strategy-name": "LEAST_CONFIDENCE",  
      "threshold": 0.4  
    }  
  },  
  "ConfigID4": {  
    "s3-bucket-name": "my-aws-bucket-entropy-sampling",  
    "content-type": "image/png",  
    "file-ext": "png",  
    "sampling-strategy": {
```

```
        "strategy-name": "ENTROPY",
        "threshold": 2
    },
    "ConfigID5": {
        "s3-bucket-name": "my-aws-bucket-no-sampling",
        "s3-prefix": "DeviceA",
        "content-type": "application/json"
    }
}
```

Samplingstrategien

Der Konnektor unterstützt vier Samplingstrategien, die bestimmen, ob Beispiele hochgeladen werden sollen, die an den Konnektor übergeben werden. Beispiele sind einzelne Dateninstanzen, die von einem Modell für eine Prognose verwendet werden. Sie können Samplingstrategien verwenden, um nach den Beispielen zu filtern, die die Modellgenauigkeit am ehesten verbessern.

RANDOM_SAMPLING

Lädt nach dem Zufallsprinzip Beispiele basierend auf der angegebenen Rate hoch. Ein Beispiel wird hochgeladen, wenn ein zufällig generierter Wert kleiner als die Rate ist. Je höher die Rate, desto mehr Beispiele werden hochgeladen.

Note

Diese Strategie ignoriert alle bereitgestellten Modellprognosen.

LEAST_CONFIDENCE

Lädt Beispiele hoch, deren maximale Zuverlässigkeitswahrscheinlichkeit unter den angegebenen Schwellenwert fällt.

Beispielszenario:

Schwellenwert: .6

Modellprognose: [.2, .2, .4, .2]

Maximale Zuverlässigkeitswahrscheinlichkeit: .4

Ergebnis:

Verwenden Sie das Beispiel, da die maximale Zuverlässigkeitswahrscheinlichkeit (.4) $\leq$ Schwellenwert (.6) ist.

MARGIN

Lädt Beispiele hoch, wenn der Abstand zwischen den beiden höchsten Zuverlässigkeitswahrscheinlichkeiten innerhalb des angegebenen Schwellenwerts liegt. Der Abstand ist die Differenz zwischen den beiden höchsten Zuverlässigkeitswahrscheinlichkeiten.

Beispielszenario:

Schwellenwert: .02

Modellprognose: [.3, .35, .34, .01]

Zwei höchste Zuverlässigkeitswahrscheinlichkeiten: [.35, .34]

Abstand: .01 (.35 - .34)

Ergebnis:

Verwenden Sie das Beispiel, da der Abstand (.01) $\leq$ Schwellenwert (.02) ist.

ENTROPY

Lädt Beispiele hoch, deren Entropie größer als der angegebene Schwellenwert ist. Verwendet die normalisierte Entropie der Modellprognose.

Beispielszenario:

Schwellenwert: 0.75

Modellprognose: [.5, .25, .25]

Entropie für die Prognose: 1.03972

Ergebnis:

Verwenden Sie das Beispiel, da die Entropie (1.03972) $>$ Schwellenwert (0.75) ist.

Eingabedaten

Benutzerdefinierte Lambda-Funktionen verwenden die `publish` Funktion des `feedback` Clients im AWS IoT Greengrass Machine Learning SDK, um den Connector aufzurufen. Ein Beispiel finden Sie unter [the section called “Beispiel für die Verwendung”](#).

 Note

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Die `publish`-Funktion verwendet die folgenden Argumente:

ConfigId

Die ID der Ziel-Feedback-Konfiguration. Dies muss mit der ID einer Feedback-Konfiguration übereinstimmen, die im [FeedbackConfigurationMap](#) Parameter für den ML-Feedback-Konnektor definiert ist.

Erforderlich: wahr

Type: Zeichenkette

ModellInput

Die Eingabedaten, die an ein Modell zur Inferenz übergeben wurden. Diese Eingabedaten werden unter Verwendung der Zielkonfiguration hochgeladen, es sei denn, sie werden basierend auf der Samplingstrategie herausgefiltert.

Erforderlich: wahr

Typ: Bytes

ModelPrediction

Die Prognoseergebnisse aus dem Modell. Ergebnistyp kann ein Wörterbuch oder eine Liste sein. Beispielsweise handelt es sich bei den Prognoseergebnissen des ML Image Classification-Konnektors um eine Liste von Wahrscheinlichkeiten (z. B. `[0.25, 0.60, 0.15]`). Diese Daten werden im Thema `/feedback/message/prediction` veröffentlicht.

Erforderlich: wahr

Typ: Wörterbuch oder `float` Werteliste

Metadaten

Kundendefinierte, anwendungsspezifische Metadaten, die an das hochgeladene Beispiel angefügt und im Thema `/feedback/message/prediction` veröffentlicht werden. Der Konnektor fügt auch einen `publish-ts`-Schlüssel mit einem Zeitstempelwert in die Metadaten ein.

Erforderlich: falsch

Typ: Wörterbuch

Beispiel: {"some-key": "some value"}

Ausgabedaten

Dieser Konnektor veröffentlicht Daten in drei MQTT-Themen:

- Statusinformationen vom Konnektor im Thema `feedback/message/status`.
- Prognoseergebnisse im Thema `feedback/message/prediction`.
- Metriken, die für das CloudWatch `cloudwatch/metric/put` Thema bestimmt sind.

Sie müssen Abonnements konfigurieren, damit der Konnektor über MQTT-Themen kommunizieren kann. Weitere Informationen finden Sie unter [the section called “Eingaben und Ausgaben”](#).

Themenfilter: `feedback/message/status`

Verwenden Sie dieses Thema, um den Status von Beispiel-Uploads und verworfenen Beispielen zu überwachen. Der Konnektor veröffentlicht jedes Mal, wenn er eine Anforderung erhält, in diesem Thema.

Beispielausgabe: Beispiel-Upload erfolgreich

```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
"1bbqaDVF0hMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
```



```

        "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDl1LWE4YmQtNzMyYzEXAMPLEoUw\"",
        "x-amz-request-id": "79104EXAMPLEB723",
        "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
        "date": "Thu, 11 Jul 2019 00:12:50 GMT",
        "x-amz-server-side-encryption": "AES256"
    }
},
    "bucket": "greengrass-feedback-connector-data-us-west-2",
    "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
    "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWI2Zi00ZDl1LWE4YmQtNzMyYzEXAMPLEoUw\"",
    "key": "s3-key-prefix/UUID.file_ext",
    "ServerSideEncryption": "AES256"
}
},
    "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}

```

Der Connector fügt die key Felder bucket und zur Antwort von Amazon S3 hinzu. Weitere Informationen zur Amazon S3 S3-Antwort finden Sie unter [PUT-Objekt](#) in der Amazon Simple Storage Service API-Referenz.

Beispielausgabe: Beispiel aufgrund der Samplingstrategie verworfen

```

{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}

```

Beispielausgabe: Beispiel-Upload fehlgeschlagen

Ein Fehlerstatus enthält die Fehlermeldung als Wert für `error_message` und die Ausnahmeklasse als Wert für `error`.

```

{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed
to upload model input data due to exception. Model prediction will not be

```

```
published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket)
when calling the PutObject operation: The specified bucket does not exist",
  "error": "NoSuchBucket"
},
"id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Beispielausgabe: Anforderung aufgrund des Anforderungslimits gedrosselt

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Themenfilter: feedback/message/prediction

Verwenden Sie dieses Thema, um auf Prognosen basierend auf hochgeladenen Daten zu achten. Auf diese Weise können Sie die Modellleistung in Echtzeit analysieren. Modellvorhersagen werden zu diesem Thema nur veröffentlicht, wenn Daten erfolgreich auf Amazon S3 hochgeladen wurden. Nachrichten, die in diesem Thema veröffentlicht werden, haben das JSON-Format. Sie enthalten den Link zu dem hochgeladenen Datenobjekt, die Prognose des Modells und die Metadaten in der Anforderung.

Sie können Prognoseergebnisse auch speichern und mithilfe dieser Daten Trends im Laufe der Zeit melden und analysieren. Trends können wertvolle Erkenntnisse liefern. Ein Trend zu abnehmender Genauigkeit im Zeitverlauf kann Ihnen beispielsweise helfen, zu entscheiden, ob das Modell neu geschult werden muss.

Beispielausgabe

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/
UUID.file_ext",
  "model-prediction": [
    0.5,
    0.2,
    0.2,
```

```
    0.1
  ],
  "config-id": "ConfigID2",
  "metadata": {
    "publish-ts": "2019-07-11 00:12:48.816752"
  }
}
```

 Tip

Sie können den [IoT Analytics Analytics-Connector](#) so konfigurieren, dass er dieses Thema abonniert und die Informationen AWS IoT Analytics zur weiteren oder historischen Analyse an diesen sendet.

Themenfilter: `cloudwatch/metric/put`

Dies ist das Ausgabethema, das zum Veröffentlichen von Metriken in CloudWatch verwendet wird. Für diese Funktion müssen Sie den [CloudWatch Metrics-Connector](#) installieren und konfigurieren.

Zu den Metriken gehören:

- Die Anzahl der hochgeladenen Beispiele.
- Die Größe der hochgeladenen Beispiele.
- Die Anzahl der Fehler bei Uploads auf Amazon S3.
- Die Anzahl der verworfenen Beispiele basierend auf der Samplingstrategie.
- Die Anzahl der gedrosselten Anforderungen.

Beispielausgabe: Größe des Datenbeispiels (veröffentlicht vor dem tatsächlichen Upload)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

Beispielausgabe: Beispiel-Upload erfolgreich

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

Beispielausgabe: Beispiel-Upload erfolgreich und Prognoseergebnis veröffentlicht

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleAndPredictionPublished"
    }
  }
}
```

Beispielausgabe: Beispiel-Upload fehlgeschlagen

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
}
```

Beispielausgabe: Beispiel aufgrund der Samplingstrategie verworfen

```
{
```

```
"request": {
  "namespace": "GreengrassFeedbackConnector",
  "metricData": {
    "value": 1,
    "unit": "Count",
    "metricName": "SampleNotUsed"
  }
}
```

Beispielausgabe: Anforderung aufgrund des Anforderungslimits gedrosselt

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

Beispiel für die Verwendung

Das folgende Beispiel ist eine benutzerdefinierte Lambda-Funktion, die das [AWS IoT Greengrass Machine Learning SDK](#) verwendet, um Daten an den ML-Feedback-Connector zu senden.

Note

Sie können das AWS IoT Greengrass Machine Learning SDK von der [AWS IoT Greengrass Download-Seite herunterladen](#).

```
import json
import logging
import os
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')
```

```
try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
    sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

Lizenzen

Der ML Feedback Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0

- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz

- [six](#)/MIT

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

Anschluss zur ML-Bildklassifizierung

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Die ML Image [Classification-Konnektoren](#) bieten einen Inferenzdienst für maschinelles Lernen (ML), der auf dem AWS IoT Greengrass Kern ausgeführt wird. Dieser lokale Inferenzdienst führt die Bildklassifizierung mithilfe eines Modells durch, das vom SageMaker KI-Bildklassifizierungsalgorithmus trainiert wurde.

Benutzerdefinierte Lambda-Funktionen verwenden das AWS IoT Greengrass Machine Learning SDK, um Inferenzanfragen an den lokalen Inferenzdienst zu senden. Der Dienst führt die Inferenz lokal aus und gibt die Wahrscheinlichkeit zurück, dass das Eingabebild zu bestimmten Kategorien gehört.

AWS IoT Greengrass stellt die folgenden Versionen dieses Connectors bereit, der für mehrere Plattformen verfügbar ist.

Version 2

Konnektor	Beschreibung und ARN
ML-Bildklassifizierung: Aarch64 JTX2	Inferenzdienst zur Bildklassifizierung für NVIDIA Jetson. TX2 Unterstützt die GPU-Beschleunigung. ARN: <code>arn:aws:greengrass: <i>region</i> :/connectors/ImageClassificationAarch64JTX2/versions/2</code>
ML-Bildklassifizierung x86_64	Bildklassifikation Inferenzdienst für x86_64 Plattformen. ARN: <code>arn:aws:greengrass: <i>region</i> :/connectors/ImageClassificationx86-64/versions/2</code>
ML-Bildklassifizierung ARMv7	Inferenzdienst zur Bildklassifizierung für ARMv7 Plattformen. ARN: <code>arn:aws:greengrass: <i>region</i> :/connectors/ImageClassificationARMv7/versions/2</code>

Version 1

Konnektor	Beschreibung und ARN
ML-Bildklassifizierung: Aarch64 JTX2	Inferenzdienst zur Bildklassifizierung für NVIDIA Jetson. TX2 Unterstützt die GPU-Beschleunigung. ARN: <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationAarch64JTX2/versions/1</code>
ML-Bildklassifizierung x86_64	Bildklassifikation Inferenzdienst für x86_64 Plattformen. ARN: <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationx86-64/versions/1</code>
ML-Bildklassifizierung Armv7	Inferenzdienst zur Bildklassifizierung für ARMv7-Plattformen. ARN: <code>arn:aws:greengrass: <i>region</i>:/connectors/ImageClassificationARMv7/versions/1</code>

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Für diese Konnektoren gelten die folgenden Anforderungen:

Version 2

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.

- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

 Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Abhängigkeiten für das auf dem Kerngerät installierte MXNet Apache-Framework. Weitere Informationen finden Sie unter [the section called “MXNet Abhängigkeiten installieren”](#).
- Eine [ML-Ressource](#) in der Greengrass-Gruppe, die auf eine SageMaker KI-Modellquelle verweist. Dieses Modell muss durch den SageMaker KI-Algorithmus zur Bildklassifizierung trainiert werden. Weitere Informationen finden Sie unter [Algorithmus zur Bildklassifizierung](#) im Amazon SageMaker AI Developer Guide.
- Der [ML Feedback-Connector](#) wurde der Greengrass-Gruppe hinzugefügt und konfiguriert. Nur erforderlich, wenn Sie den Konnektor verwenden möchten, um Modelleingabedaten hochzuladen und Prognosen in einem MQTT-Thema zu veröffentlichen.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `sagemaker:DescribeTrainingJob` Aktion für den Ziel-Trainingsjob zulässt, wie im folgenden Beispiel für eine IAM-Richtlinie gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:DescribeTrainingJob"
      ],
      "Resource": "arn:aws:sagemaker:us-east-1:123456789012:training-job/training-job-name"
    }
  ]
}
```

```
}  
  ]  
}
```

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *). Wenn Sie den Zielausbildungsjob in future ändern, stellen Sie sicher, dass Sie die Gruppenrolle aktualisieren.

- AWS IoT Greengrass Für die Interaktion mit diesem Connector ist das [Machine Learning SDK v1.1.0](#) erforderlich.

Version 1

- AWS IoT Greengrass Core Software v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Abhängigkeiten für das auf dem Kerngerät installierte MXNet Apache-Framework. Weitere Informationen finden Sie unter [the section called “MXNet Abhängigkeiten installieren”](#).
- Eine [ML-Ressource](#) in der Greengrass-Gruppe, die auf eine SageMaker KI-Modellquelle verweist. Dieses Modell muss durch den SageMaker KI-Algorithmus zur Bildklassifizierung trainiert werden. Weitere Informationen finden Sie unter [Algorithmus zur Bildklassifizierung](#) im Amazon SageMaker AI Developer Guide.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `sagemaker:DescribeTrainingJob` Aktion für den Ziel-Trainingsjob zulässt, wie im folgenden Beispiel für eine IAM-Richtlinie gezeigt.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  

```

```
        "sagemaker:DescribeTrainingJob"  
    ],  
    "Resource": "arn:aws:sagemaker:us-east-1:123456789012:training-  
job/training-job-name"  
  }  
]  
}
```

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *). Wenn Sie den Zielausbildungsjob in future ändern, stellen Sie sicher, dass Sie die Gruppenrolle aktualisieren.

- AWS IoT Greengrass Für die Interaktion mit diesem Connector ist [Machine Learning SDK v1.0.0](#) oder höher erforderlich.

Konnektor-Parameter

Diese Konnektoren stellen die folgenden Parameter bereit.

Version 2

MLModelDestinationPath

Der absolute lokale Pfad der ML-Ressource innerhalb der Lambda-Umgebung. Dies ist der Zielpfad, der für die ML-Ressource angegeben ist.

Note

Wenn Sie die ML-Ressource in der Konsole angelegt haben, ist dies der lokale Pfad.

Anzeigenname in der AWS IoT Konsole: Zielpfad des Modells

Erforderlich: true

Typ: string

Gültiges Muster: .+

MLModelResourceId

Die ID der ML-Ressource, die auf das Quellmodell verweist.

Anzeigename in der AWS IoT Konsole: SageMaker Job-ARN-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: [a-zA-Z0-9: _-]+

MLModelSageMakerJobArn

Der ARN des SageMaker KI-Trainingsjobs, der die SageMaker KI-Modellquelle darstellt. Das Modell muss mit dem SageMaker KI-Bildklassifizierungsalgorithmus trainiert werden.

Anzeigename in der AWS IoT Konsole: SageMaker Job ARN

Erforderlich: true

Typ: string

Gültiges Muster: ^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+\$

LocalInferenceServiceName

Der Name für den lokalen Inferenzdienst. Benutzerdefinierte Lambda-Funktionen rufen den Dienst auf, indem sie den Namen an die `invoke_inference_service` Funktion des AWS IoT Greengrass Machine Learning SDK übergeben. Ein Beispiel finden Sie unter [the section called "Beispiel für eine Verwendung"](#).

Anzeigename in der AWS IoT Konsole: Name des lokalen Inferenzdienstes

Erforderlich: true

Typ: string

Gültiges Muster: [a-zA-Z0-9][a-zA-Z0-9-]{1,62}

LocalInferenceServiceTimeoutSeconds

Die Zeitspanne (in Sekunden), nach der die Inferenzanforderung beendet wird. Der minimale Wert beträgt 1.

Anzeigename in der AWS IoT Konsole: Timeout (Sekunde)

Erforderlich: true

Typ: string

Gültiges Muster: [1-9][0-9]*

LocalInferenceServiceMemoryLimitKB

Die Speichergröße (in KB), auf die der Service Zugriff hat. Der minimale Wert beträgt 1.

Anzeigename in der AWS IoT Konsole: Speicherlimit (KB)

Erforderlich: true

Typ: string

Gültiges Muster: [1-9][0-9]*

GPUAcceleration

Der CPU- oder GPU-(beschleunigte) Berechnungskontext. Diese Eigenschaft gilt nur für den JTX2 Aarch64-Konnektor von ML Image Classification.

Anzeigename in der AWS IoT Konsole: GPU-Beschleunigung

Erforderlich: true

Typ: string

Gültige Werte: CPU oder GPU.

MLFeedbackConnectorConfigId

Die ID der Feedback-Konfiguration, die zum Hochladen von Modelleingabedaten verwendet werden soll. Diese muss mit der ID einer Feedback-Konfiguration übereinstimmen, die für den [ML-Feedback-Konnektor](#) definiert ist.

Dieser Parameter ist nur erforderlich, wenn Sie den ML-Feedback-Konnektor verwenden möchten, um Modelleingabedaten hochzuladen und Prognosen in einem MQTT-Thema zu veröffentlichen.

Anzeigename in der AWS IoT Konsole: Konfigurations-ID des ML Feedback-Connectors

Erforderlich: false

Typ: string

Gültiges Muster: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Version 1

MLModelDestinationPath

Der absolute lokale Pfad der ML-Ressource innerhalb der Lambda-Umgebung. Dies ist der Zielpfad, der für die ML-Ressource angegeben ist.

Note

Wenn Sie die ML-Ressource in der Konsole angelegt haben, ist dies der lokale Pfad.

Anzeigename in der AWS IoT Konsole: Zielpfad des Modells

Erforderlich: true

Typ: string

Gültiges Muster: `.+`

MLModelResourceId

Die ID der ML-Ressource, die auf das Quellmodell verweist.

Anzeigename in der AWS IoT Konsole: SageMaker Job-ARN-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

Der ARN des SageMaker KI-Trainingsjobs, der die SageMaker KI-Modellquelle darstellt. Das Modell muss mit dem SageMaker KI-Bildklassifizierungsalgorithmus trainiert werden.

Anzeigename in der AWS IoT Konsole: SageMaker Job ARN

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

Der Name für den lokalen Inferenzdienst. Benutzerdefinierte Lambda-Funktionen rufen den Dienst auf, indem sie den Namen an die `invoke_inference_service` Funktion des AWS IoT Greengrass Machine Learning SDK übergeben. Ein Beispiel finden Sie unter [the section called “Beispiel für eine Verwendung”](#).

Anzeigename in der AWS IoT Konsole: Name des lokalen Inferenzdienstes

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

Die Zeitspanne (in Sekunden), nach der die Inferenzanforderung beendet wird. Der minimale Wert beträgt 1.

Anzeigename in der AWS IoT Konsole: Timeout (Sekunde)

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `[1-9][0-9]*`

LocalInferenceServiceMemoryLimitKB

Die Speichergröße (in KB), auf die der Service Zugriff hat. Der minimale Wert beträgt 1.

Anzeigename in der AWS IoT Konsole: Speicherlimit (KB)

Erforderlich: true

Typ: string

Gültiges Muster: [1-9][0-9]*

GPUAcceleration

Der CPU- oder GPU-(beschleunigte) Berechnungskontext. Diese Eigenschaft gilt nur für den JTX2 Aarch64-Konnektor von ML Image Classification.

Anzeigename in der AWS IoT Konsole: GPU-Beschleunigung

Erforderlich: true

Typ: string

Gültige Werte: CPU oder GPU.

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Die folgenden CLI-Befehle erstellen eine `ConnectorDefinition` mit einer ersten Version, die einen ML Image Classification Connector enthält.

Example: CPU Instance (Beispiel: CPU-Instance)

In diesem Beispiel wird eine Instanz des ML Image Classification ARMv7L-Connectors erstellt.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyImageClassificationConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ImageClassificationARMv7/versions/2",  
      "Parameters": {  
        "MLModelDestinationPath": "/path-to-model",  
        "MLModelResourceId": "my-ml-resource",
```

```

        "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
        "LocalInferenceServiceName": "imageClassification",
        "LocalInferenceServiceTimeoutSeconds": "10",
        "LocalInferenceServiceMemoryLimitKB": "500000",
        "MLFeedbackConnectorConfigId": "MyConfig0"
    }
}
]
}'

```

Example: GPU Instance (Beispiel: GPU-Instance)

In diesem Beispiel wird eine Instanz des ML Image Classification JTX2 Aarch64-Connectors erstellt, der die GPU-Beschleunigung auf einem NVIDIA-Jetson-Board unterstützt. TX2

```

aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationAarch64JTX2/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "GPUAcceleration": "GPU",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'

```

Note

Die Lambda-Funktion in diesen Steckverbindern hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Diese Konnektoren akzeptieren eine Bilddatei als Eingabe. Eingabebilddateien müssen im jpeg- oder png-Format vorliegen. Weitere Informationen finden Sie unter [the section called “Beispiel für eine Verwendung”](#).

Diese Konnektoren akzeptieren keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Diese Konnektoren geben eine formatierte Prognose für das im Eingabebild identifizierte Objekt zurück:

```
[0.3,0.1,0.04,...]
```

Die Prognose enthält eine Liste von Werten, die den Kategorien entsprechen, die während der Modellschulung im Schulungs-Dataset verwendet werden. Jeder Wert stellt die Wahrscheinlichkeit dar, dass das Bild unter die entsprechende Kategorie fällt. Die Kategorie mit der höchsten Wahrscheinlichkeit ist die dominante Prognose.

Diese Konnektoren veröffentlichen keine MQTT-Nachrichten als Ausgabedaten.

Beispiel für eine Verwendung

Die folgende Lambda-Beispielfunktion verwendet das [AWS IoT Greengrass Machine Learning SDK](#), um mit einem ML Image Classification Connector zu interagieren.

Note

Sie können das SDK von der Downloadseite des [AWS IoT Greengrass Machine Learning SDK](#) herunterladen.

Das Beispiel initialisiert einen SDK-Client und ruft synchron die Funktion `invoke_inference_service` des SDK auf, um den lokalen Inferenzdienst aufzurufen. Es wird der Algorithmustyp, der Servicename, der Bildtyp und der Bildinhalt übergeben. Anschließend analysiert das Beispiel die Service-Antwort, um die Wahrscheinlichkeitsergebnisse (Vorhersagen) zu erhalten.

Python 3.7

```

import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}'.format(e.__class__.__name__,
e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')

```

```

    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()

def function_handler(event, context):
    return

```

Python 2.7

```

import logging
from threading import Timer

import numpy

import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")

def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)

```

```
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    predictions_arr = numpy.fromstring(predictions, sep=",")
    logging.info("Split into %s predictions.", len(predictions_arr))

    # Perform business logic that relies on predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()

infer()

# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return
```

Die `invoke_inference_service` Funktion im AWS IoT Greengrass Machine Learning SDK akzeptiert die folgenden Argumente.

Argument	Description
AlgoType	Der Name des Algorithmentyps, der für die Inferenz verwendet werden soll. Derzeit wird nur <code>image-classification</code> unterstützt. Erforderlich: <code>true</code> Typ: <code>string</code>

Argument	Description
	Zulässige Werte: <code>image-classification</code>
ServiceName	Der Name des lokalen Inferenzdienstes. Verwenden Sie den Namen, den Sie für den Parameter <code>LocalInferenceServiceName</code> bei der Konfiguration des Konnektors angegeben haben. Erforderlich: <code>true</code> Typ: <code>string</code>
ContentType	Der Mime-Typ des Eingangsbildes. Erforderlich: <code>true</code> Typ: <code>string</code> Zulässige Werte: <code>image/jpeg</code>, <code>image/png</code>
Body	Der Inhalt der Eingabebilddatei. Erforderlich: <code>true</code> Typ: <code>binary</code>

MXNet Abhängigkeiten auf dem AWS IoT Greengrass Core installieren

Um einen ML Image Classification Connector zu verwenden, müssen Sie die Abhängigkeiten für das MXNet Apache-Framework auf dem Kerngerät installieren. Die Konnektoren verwenden das Framework für die Bereitstellung des ML-Modells.

Note

Diese Konnektoren sind mit einer vorkompilierten MXNet Bibliothek gebündelt, sodass Sie das MXNet Framework nicht auf dem Kerngerät installieren müssen.

AWS IoT Greengrass bietet Skripte zur Installation der Abhängigkeiten für die folgenden gängigen Plattformen und Geräte (oder zur Verwendung als Referenz für deren Installation). Wenn Sie eine andere Plattform oder ein anderes Gerät verwenden, finden Sie in der [MXNet Dokumentation](#) zu Ihrer Konfiguration weitere Informationen.

Stellen Sie vor der Installation der MXNet Abhängigkeiten sicher, dass die erforderlichen [Systembibliotheken](#) (mit den angegebenen Mindestversionen) auf dem Gerät vorhanden sind.

NVIDIA Jetson TX2

1. Installieren Sie CUDA Toolkit 9.0 und cuDNN 7.0. Befolgen Sie die Anweisungen unter [the section called "Einrichten anderer Geräte"](#) im Tutorial "Erste Schritte".
2. Aktivieren Sie Universum-Repositorys, damit der Konnektor offene, von der Community verwaltete Software installieren kann. Weitere Informationen finden Sie unter [Repositories/Ubuntu](#) in der Ubuntu-Dokumentation.
 - a. Öffnen Sie die `/etc/apt/sources.list` Datei.
 - b. Achten Sie darauf, dass die folgenden Zeilen nicht kommentiert sind.

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

3. Speichern Sie eine Kopie des folgenden Installationsskripts in einer Datei namens `nvidiajtx2.sh` auf dem Core-Gerät.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
```



```
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

Wenn [OpenCV](#) mit diesem Skript nicht erfolgreich installiert wird, können Sie versuchen, es aus der Quelle zu erstellen. Weitere Informationen finden Sie unter [Installation in Linux](#) in der OpenCV-Dokumentation oder in anderen Online-Ressourcen für Ihre Plattform.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
python-dev

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. Führen Sie in dem Verzeichnis, in dem Sie die Datei gespeichert haben, den folgenden Befehl aus:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Speichern Sie eine Kopie des folgenden Installationsskripts in einer Datei namens `x86_64.sh` auf dem Core-Gerät.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == "Ubuntu" ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
    apt-get install -y python3.7 python3.7-dev
elif [ "$release" == "Amazon Linux" ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
    echo "OS Release not supported: $release"
    exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

Wenn [OpenCV](#) mit diesem Skript nicht erfolgreich installiert wird, können Sie versuchen, es aus der Quelle zu erstellen. Weitere Informationen finden Sie unter [Installation in Linux](#) in der OpenCV-Dokumentation oder in anderen Online-Ressourcen für Ihre Plattform.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
    python-pip
elif [ "$release" == '"Amazon Linux"' ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
    pip
else
    echo "OS Release not supported: $release"
    exit 1
fi

pip install numpy==1.15.0 scipy opencv-python
```

```
echo 'Dependency installation/upgrade complete.'
```

2. Führen Sie in dem Verzeichnis, in dem Sie die Datei gespeichert haben, den folgenden Befehl aus:

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. Speichern Sie eine Kopie des folgenden Installationsskripts in einer Datei namens `armv7l.sh` auf dem Core-Gerät.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
OpenCV with pip on this platform. Try building the latest OpenCV from source
(https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

Note

Wenn [OpenCV](#) mit diesem Skript nicht erfolgreich installiert wird, können Sie versuchen, es aus der Quelle zu erstellen. Weitere Informationen finden Sie unter [Installation in Linux](#) in der OpenCV-Dokumentation oder in anderen Online-Ressourcen für Ihre Plattform.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
# repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
# installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. Führen Sie in dem Verzeichnis, in dem Sie die Datei gespeichert haben, den folgenden Befehl aus:

```
sudo bash armv7l.sh
```

Note

Auf einem Raspberry Pi ist die Installation von Machine Learning-Abhängigkeiten mithilfe von pip eine speicherintensive Operation, die dazu führen kann, dass das Gerät nicht mehr genügend Arbeitsspeicher hat und nicht mehr reagiert. Um dieses Problem zu umgehen, können Sie die Auslagerungsgröße vorübergehend erhöhen:

Erhöhen Sie in `/etc/dphys-swapfile` den Wert der `CONF_SWAPSIZE`-Variable und führen Sie dann den folgenden Befehl aus, um `dphys-swapfile` neu zu starten.

```
/etc/init.d/dphys-swapfile restart
```

Protokollierung und Problemlösungen

Abhängig von Ihren Gruppeneinstellungen werden Ereignis- und Fehlerprotokolle in CloudWatch Logs, in das lokale Dateisystem oder in beide geschrieben. Protokolle von diesem Konnektor verwenden das Präfix `LocalInferenceServiceName`. Wenn sich der Konnektor nicht wie erwartet verhält, überprüfen Sie die Protokolle des Konnektors. Diese enthalten in der Regel nützliche Debugging-Informationen, wie z. B. eine fehlende ML-Bibliothek-Abhängigkeit oder die Ursache eines Fehlers beim Starten des Konnektors.

Wenn die AWS IoT Greengrass Gruppe so konfiguriert ist, dass sie lokale Protokolle schreibt, schreibt der Connector Protokolldateien in `greengrass-root/ggc/var/log/user/region/aws/`. Weitere Hinweise zur Greengrass-Protokollierung finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

Verwenden Sie die folgenden Informationen, um Probleme mit den ML Image Classification-Konnektoren zu beheben.

Erforderliche Systembibliotheken

Auf den folgenden Registerkarten sind die Systembibliotheken aufgeführt, die für jeden ML Image Classification-Konnektor erforderlich sind.

ML Image Classification Aarch64 JTX2

Bibliothek	Mindestversion
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	nicht zutreffend

Bibliothek	Mindestversion
libcudart.so.9.0	nicht zutreffend
libcudnn.so.7	nicht zutreffend
libcufft.so.9.0	nicht zutreffend
libcurand.so.9.0	nicht zutreffend
libcusolver.so.9.0	nicht zutreffend
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Image Classification x86_64

Bibliothek	Mindestversion
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5

Bibliothek	Mindestversion
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

Bibliothek	Mindestversion
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problembereiche

Symptom	Lösung
Auf einem Raspberry Pi wird die folgende Fehlermeldung protokolliert, und Sie verwenden nicht die Kamera: <code>Failed to initialize libdc1394</code>	Führen Sie den folgenden Befehl aus, um den Treiber zu deaktivieren: <pre>sudo ln /dev/null /dev/raw1394</pre> Diese Operation ist flüchtig, und der symbolische Link wird nach dem Neustart verschwinden. Schlagen Sie im Handbuch Ihrer Betriebs

Symptom	Lösung
	ystemverteilung nach, wie der Link beim Neustart automatisch erstellt wird.

Lizenzen

Die ML Image Classification Connectors beinhalten die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz
- [Deep Neural Network Library \(DNNL\)](#)/Apache-Lizenz 2.0
- [OpenMP* Runtime Library](#)/Siehe [Lizenzierung der Intel OpenMP Runtime Library](#).
- [mxnet](#)/Apache-Lizenz 2.0
- [six](#)/MIT

Lizenzierung der Intel OpenMP Runtime Library. Die Intel® OpenMP* Runtime ist zweifach lizenziert, mit einer kommerziellen (COM) -Lizenz als Teil der Intel® Parallel Studio XE Suite-Produkte und einer BSD-Open-Source-Lizenz (OSS).

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
2	Der <code>MLFeedbackConnectorConfigId</code> Parameter wurde hinzugefügt, um die Verwendung des ML-Feedback-Konnektors zum Hochladen von Modelleingabedaten, zum Veröffentlichen von Vorhersagen zu einem MQTT-Thema und zum Veröffentlichen von Metriken auf Amazon CloudWatch zu unterstützen.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Durchführen von Machine Learning-Inferenzen](#)
- [Algorithmus zur Bildklassifizierung](#) im Amazon SageMaker AI Developer Guide

Anschluss zur ML-Objekterkennung

Warning

Dieser Konnektor befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Die [ML-Objekterkennungs-Konnektoren](#) bieten einen Inferenzdienst für maschinelles Lernen (ML), der auf dem AWS IoT Greengrass Kern ausgeführt wird. Dieser lokale Inferenzdienst führt die Objekterkennung mithilfe eines Objekterkennungsmodells durch, das vom SageMaker AI Neo Deep Learning-Compiler kompiliert wurde. Es werden zwei Arten von Objekterkennungsmodellen unterstützt: Single Shot Multibox Detector (SSD) und You Only Look Once (YOLO) v3. Weitere Informationen finden Sie unter [Objekterkennungsmodell-Anforderungen](#).

Benutzerdefinierte Lambda-Funktionen verwenden das AWS IoT Greengrass Machine Learning SDK, um Inferenzanfragen an den lokalen Inferenzdienst zu senden. Der Service führt eine lokale Inferenz für ein Eingabebild aus und gibt eine Liste mit Voraussagen für jedes im Bild erkannte Objekt zurück. Jede Voraussage enthält eine Objektkategorie, einen Zuverlässigkeitswert für Voraussagen und Pixelkoordinaten, die einen Begrenzungsrahmen um das vorhergesagte Objekt angeben.

AWS IoT Greengrass bietet Konnektoren zur ML-Objekterkennung für mehrere Plattformen:

Konnektor	Beschreibung und ARN
ML-Objekterkennung (Aarch64) JTX2	Inferenzdienst zur Objekterkennung für NVIDIA Jetson. TX2 Unterstützt die GPU-Beschleunigung. ARN: <code>arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionAarch64JTX2/versions/1</code>
ML-Objekterkennung x86_64	Inferenzservice für die Objekterkennung für x86_64-Plattformen. ARN: <code>arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionx86-64/versions/1</code>
ML-Objekterkennung ARMv7	Inferenzdienst zur Objekterkennung für ARMv7 Plattformen. ARN: <code>arn:aws:greengrass: <i>region</i>::/connectors/ObjectDetectionARMv7/versions/1</code>

Voraussetzungen

Für diese Konnektoren gelten die folgenden Anforderungen:

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Abhängigkeiten für die SageMaker AI Neo Deep Learning-Runtime, die auf dem Kerngerät installiert ist. Weitere Informationen finden Sie unter [the section called “Installieren von Neo Deep Learning Runtime-Abhängigkeiten”](#).
- Eine [ML-Ressource](#) in der Greengrass-Gruppe. Die ML-Ressource muss auf einen Amazon S3 S3-Bucket verweisen, der ein Objekterkennungsmodell enthält. Weitere Informationen finden Sie unter [Amazon S3 S3-Modellquellen](#).

Note

Das Modell muss ein Objekterkennungsmodell des Typs Single Shot Multibox Detector oder You Only Look Once v3 sein. Es muss mit dem SageMaker AI Neo Deep Learning-Compiler kompiliert werden. Weitere Informationen finden Sie unter [Objekterkennungsmodell-Anforderungen](#).

- Der [ML Feedback-Connector](#) wurde der Greengrass-Gruppe hinzugefügt und konfiguriert. Nur erforderlich, wenn Sie den Konnektor verwenden möchten, um Modelleingabedaten hochzuladen und Prognosen in einem MQTT-Thema zu veröffentlichen.
- AWS IoT Greengrass Für die Interaktion mit diesem Connector ist das [Machine Learning SDK](#) v1.1.0 erforderlich.

Objekterkennungsmodell-Anforderungen

Die ML-Objekterkennungsanschlüsse unterstützen die Objekterkennungsmodelle Single Shot Multibox Detector (SSD) und You Only Look Once (YOLO) v3. Sie können die von [GluonCV](#) bereitgestellten Objekterkennungskomponenten verwenden, um das Modell mit Ihrem eigenen Datensatz zu trainieren. Sie können auch vorab trainierte Modelle aus dem GluonCV Model Zoo verwenden:

- [Vortrainiertes SSD-Modell](#)
- [Vortrainiertes YOLO v3-Modell](#)

Ihr Objekterkennungsmodell muss mit 512-x-512-Eingabebildern trainiert worden sein. Die vortrainierten Modelle aus dem GluonCV Model Zoo erfüllen diese Anforderung bereits.

Trainierte Objekterkennungsmodelle müssen mit dem SageMaker AI Neo Deep Learning-Compiler kompiliert werden. Stellen Sie beim Kompilieren sicher, dass die Zielhardware mit der Hardware Ihres Greengrass Core-Geräts übereinstimmt. Weitere Informationen finden Sie unter [SageMaker AI Neo](#) im Amazon SageMaker AI Developer Guide.

Das kompilierte Modell muss als ML-Ressource ([Amazon S3 S3-Modellquelle](#)) derselben Greengrass-Gruppe wie der Connector hinzugefügt werden.

Konnektor-Parameter

Diese Konnektoren stellen die folgenden Parameter bereit.

MLModelDestinationPath

Der absolute Pfad zum Amazon S3 S3-Bucket, der das NEO-kompatible ML-Modell enthält. Dies ist der Zielpfad, der für die ML-Modellressource angegeben ist.

Anzeigenname in der AWS IoT Konsole: Zielpfad des Modells

Erforderlich: true

Typ: string

Gültiges Muster: .+

MLModelResourceId

Die ID der ML-Ressource, die auf das Quellmodell verweist.

Anzeigename in der AWS IoT Konsole: Greengrass-Gruppen-ML-Ressource

Erforderlich: true

Typ: S3MachineLearningModelResource

Gültiges Muster: `^[a-zA-Z0-9:_-]+$`

LocalInferenceServiceName

Der Name für den lokalen Inferenzdienst. Benutzerdefinierte Lambda-Funktionen rufen den Dienst auf, indem sie den Namen an die `invoke_inference_service` Funktion des AWS IoT Greengrass Machine Learning SDK übergeben. Ein Beispiel finden Sie unter [the section called "Beispiel für eine Verwendung"](#).

Anzeigename in der AWS IoT Konsole: Name des lokalen Inferenzdienstes

Erforderlich: true

Typ: string

Gültiges Muster: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

LocalInferenceServiceTimeoutSeconds

Die Zeitspanne (in Sekunden), nach der die Inferenzanforderung beendet wird. Der minimale Wert beträgt 1. Der Standardwert lautet 10.

Anzeigename in der AWS IoT Konsole: Timeout (Sekunde)

Erforderlich: true

Typ: string

Gültiges Muster: `^[1-9][0-9]*$`

LocalInferenceServiceMemoryLimitKB

Die Speichergröße (in KB), auf die der Service Zugriff hat. Der minimale Wert beträgt 1.

Anzeigename in der AWS IoT Konsole: Speicherlimit

Erforderlich: true

Typ: string

Gültiges Muster: `^[1-9][0-9]*$`

GPUAcceleration

Der CPU- oder GPU-(beschleunigte) Berechnungskontext. Diese Eigenschaft gilt nur für den JTX2 Aarch64-Konnektor von ML Image Classification.

Anzeigenname in der AWS IoT Konsole: GPU-Beschleunigung

Erforderlich: `true`

Typ: `string`

Gültige Werte: CPU oder GPU.

MLFeedbackConnectorConfigId

Die ID der Feedback-Konfiguration, die zum Hochladen von Modelleingabedaten verwendet werden soll. Diese muss mit der ID einer Feedback-Konfiguration übereinstimmen, die für den [ML-Feedback-Konnektor](#) definiert ist.

Dieser Parameter ist nur erforderlich, wenn Sie den ML-Feedback-Konnektor verwenden möchten, um Modelleingabedaten hochzuladen und Prognosen in einem MQTT-Thema zu veröffentlichen.

Anzeigenname in der AWS IoT Konsole: Konfigurations-ID des ML Feedback-Connectors

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die einen ML-Objekterkennungs-Konnektor enthält. In diesem Beispiel wird eine Instanz des ML Object ARMv7l Detection-Konnektors erstellt.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyObjectDetectionConnector",
```

```

    "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ObjectDetectionARMv7/versions/1",
    "Parameters": {
        "MLModelDestinationPath": "/path-to-model",
        "MLModelResourceId": "my-ml-resource",
        "LocalInferenceServiceName": "objectDetection",
        "LocalInferenceServiceTimeoutSeconds": "10",
        "LocalInferenceServiceMemoryLimitKB": "500000",
        "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"
    }
}
]
}'

```

Note

Die Lambda-Funktion in diesen Steckverbindern hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Diese Konnektoren akzeptieren eine Bilddatei als Eingabe. Eingabebilddateien müssen im jpeg- oder png-Format vorliegen. Weitere Informationen finden Sie unter [the section called “Beispiel für eine Verwendung”](#).

Diese Konnektoren akzeptieren keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Diese Konnektoren geben eine formatierte Liste der Voraussageergebnisse für die identifizierten Objekte im Eingabebild zurück:

```

{
  "prediction": [
    [
      14,
      0.9384938478469849,

```



```

        0.37763649225234985,
        0.5110225081443787,
        0.6697432398796082,
        0.8544386029243469
    ],
    [
        14,
        0.8859519958496094,
        0,
        0.43536216020584106,
        0.3314110040664673,
        0.9538808465003967
    ],
    [
        12,
        0.04128098487854004,
        0.5976729989051819,
        0.5747185945510864,
        0.704264223575592,
        0.857937216758728
    ],
    ...
]
}

```

Jede Voraussage in der Liste wird von eckigen Klammern umschlossen und enthält sechs Werte:

- Der erste Wert stellt die vorausgesagte Objektkategorie für das identifizierte Objekt dar. Objektkategorien und ihre entsprechenden Werte werden beim Training Ihres Machine-Learning-Modells zur Objekterkennung im Neo Deep Learning-Compiler bestimmt.
- Der zweite Wert ist der Zuverlässigkeitswert für die Objektkategorievoraussage. Dieser Wert stellt die Wahrscheinlichkeit dar, mit der die Voraussage korrekt ist.
- Die letzten vier Werte entsprechen den Pixeldimensionen, die einen Begrenzungsrahmen um das vorausgesagte Objekt im Bild darstellen.

Diese Konnektoren veröffentlichen keine MQTT-Nachrichten als Ausgabedaten.

Beispiel für eine Verwendung

Die folgende Lambda-Beispielfunktion verwendet das [AWS IoT Greengrass Machine Learning SDK](#), um mit einem ML Object Detection-Konnektor zu interagieren.

 Note

Sie können das SDK von der Downloadseite des [AWS IoT Greengrass Machine Learning SDK](#) herunterladen.

Das Beispiel initialisiert einen SDK-Client und ruft synchron die Funktion `invoke_inference_service` des SDK auf, um den lokalen Inferenzdienst aufzurufen. Es wird der Algorithmustyp, der Servicename, der Bildtyp und der Bildinhalt übergeben. Anschließend analysiert das Beispiel die Service-Antwort, um die Wahrscheinlichkeitsergebnisse (Vorhersagen) zu erhalten.

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}'.format(e.__class__.__name__, e))
        return
```

```
logging.info('resp: {}'.format(resp))
predictions = resp['Body'].read().decode("utf-8")
logging.info('predictions: {}'.format(predictions))
predictions = eval(predictions)

# Perform business logic that relies on the predictions.

# Schedule the infer() function to run again in ten second.
Timer(10, infer).start()
return

infer()

def function_handler(event, context):
    return
```

Die `invoke_inference_service` Funktion im AWS IoT Greengrass Machine Learning SDK akzeptiert die folgenden Argumente.

Argument	Beschreibung
AlgoType	Der Name des Algorithmentyps, der für die Inferenz verwendet werden soll. Derzeit wird nur <code>object-detection</code> unterstützt. Erforderlich: <code>true</code> Typ: <code>string</code> Zulässige Werte: <code>object-detection</code>
ServiceName	Der Name des lokalen Inferenzdienstes. Verwenden Sie den Namen, den Sie für den Parameter <code>LocalInferenceServiceName</code> bei der Konfiguration des Konnektors angegeben haben. Erforderlich: <code>true</code> Typ: <code>string</code>

Argument	Beschreibung
ContentType	Der Mime-Typ des Eingangsbildes. Erforderlich: <code>true</code> Typ: <code>string</code> Zulässige Werte: <code>image/jpeg</code>, <code>image/png</code>
Body	Der Inhalt der Eingabebilddatei. Erforderlich: <code>true</code> Typ: <code>binary</code>

Installation der Deep-Learning-Laufzeitabhängigkeiten von Neo auf dem AWS IoT Greengrass Core

Die ML Object Detection-Konnektoren sind im Lieferumfang der SageMaker AI Neo Deep Learning Runtime (DLR) enthalten. Die Konnektoren verwenden die Laufzeit, um das ML-Modell zu bedienen. Um diese Konnektoren zu verwenden, müssen Sie die Abhängigkeiten für die DLR auf Ihrem Core-Gerät installieren.

Stellen Sie vor der Installation der DLR-Abhängigkeiten sicher, dass die erforderlichen [Systembibliotheken](#) (mit den angegebenen Mindestversionen) auf dem Gerät vorhanden sind.

NVIDIA Jetson TX2

1. Installieren Sie CUDA Toolkit 9.0 und cuDNN 7.0. Befolgen Sie die Anweisungen unter [the section called "Einrichten anderer Geräte"](#) im Tutorial "Erste Schritte".
2. Aktivieren Sie Universum-Repositorys, damit der Konnektor offene, von der Community verwaltete Software installieren kann. Weitere Informationen finden Sie unter [Repositories/Ubuntu](#) in der Ubuntu-Dokumentation.
 - a. Öffnen Sie die `/etc/apt/sources.list` Datei.
 - b. Achten Sie darauf, dass die folgenden Zeilen nicht kommentiert sind.

```
deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
```

- Speichern Sie eine Kopie des folgenden Installationsskripts in einer Datei namens `nvidiajtx2.sh` auf dem Core-Gerät.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
  with pip on this platform. Try building the latest OpenCV from source (https://
  github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

Wenn [OpenCV](#) mit diesem Skript nicht erfolgreich installiert wird, können Sie versuchen, es aus der Quelle zu erstellen. Weitere Informationen finden Sie unter [Installation in Linux](#) in der OpenCV-Dokumentation oder in anderen Online-Ressourcen für Ihre Plattform.

- Führen Sie in dem Verzeichnis, in dem Sie die Datei gespeichert haben, den folgenden Befehl aus:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Speichern Sie eine Kopie des folgenden Installationsskripts in einer Datei namens `x86_64.sh` auf dem Core-Gerät.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
    # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
    # installed, so
    # this is mostly to prepare dependencies on Ubuntu EC2 instance.
    apt-get -y update
    apt-get -y dist-upgrade

    apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
    apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
    # Amazon Linux. Expect python to be installed already
    yum -y update
    yum -y upgrade

    yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
    echo "OS Release not supported: $release"
    exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

Wenn [OpenCV](#) mit diesem Skript nicht erfolgreich installiert wird, können Sie versuchen, es aus der Quelle zu erstellen. Weitere Informationen finden Sie unter [Installation in Linux](#) in der OpenCV-Dokumentation oder in anderen Online-Ressourcen für Ihre Plattform.

2. Führen Sie in dem Verzeichnis, in dem Sie die Datei gespeichert haben, den folgenden Befehl aus:

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. Speichern Sie eine Kopie des folgenden Installationsskripts in einer Datei namens `armv7l.sh` auf dem Core-Gerät.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

 Note

Wenn [OpenCV](#) mit diesem Skript nicht erfolgreich installiert wird, können Sie versuchen, es aus der Quelle zu erstellen. Weitere Informationen finden Sie unter [Installation in Linux](#) in der OpenCV-Dokumentation oder in anderen Online-Ressourcen für Ihre Plattform.

2. Führen Sie in dem Verzeichnis, in dem Sie die Datei gespeichert haben, den folgenden Befehl aus:

```
sudo bash armv7l.sh
```

 Note

Auf einem Raspberry Pi ist die Installation von Machine Learning-Abhängigkeiten mithilfe von pip eine speicherintensive Operation, die dazu führen kann, dass das Gerät nicht mehr genügend Arbeitsspeicher hat und nicht mehr reagiert. Um dieses Problem zu umgehen, können Sie vorübergehend die Auslagerungsgröße erhöhen. Erhöhen Sie in `/etc/dphys-swapfile` den Wert der `CONF_SWAPSIZE`-Variable und führen Sie dann den folgenden Befehl aus, um `dphys-swapfile` neu zu starten.

```
/etc/init.d/dphys-swapfile restart
```

Protokollierung und Problembehandlung

Abhängig von Ihren Gruppeneinstellungen werden Ereignis- und Fehlerprotokolle in CloudWatch Logs, in das lokale Dateisystem oder in beide geschrieben. Protokolle von diesem Konnektor verwenden das Präfix `LocalInferenceServiceName`. Wenn sich der Konnektor nicht wie erwartet verhält, überprüfen Sie die Protokolle des Konnektors. Diese enthalten in der Regel nützliche Debugging-Informationen, wie z. B. eine fehlende ML-Bibliothek-Abhängigkeit oder die Ursache eines Fehlers beim Starten des Konnektors.

Wenn die AWS IoT Greengrass Gruppe so konfiguriert ist, dass sie lokale Protokolle schreibt, schreibt der Connector Protokolldateien in `greengrass-root/ggc/var/log/user/region/`

aws/. Weitere Hinweise zur Greengrass-Protokollierung finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

Verwenden Sie die folgenden Informationen, um Probleme mit den ML-Objekterkennungs-Konnektoren zu beheben.

Erforderliche Systembibliotheken

Auf den folgenden Registerkarten sind die Systembibliotheken aufgeführt, die für jeden ML Object Detection-Konnektor erforderlich sind.

ML Object Detection Aarch64 JTX2

Bibliothek	Mindestversion
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	nicht zutreffend
libcudart.so.9.0	nicht zutreffend
libcudnn.so.7	nicht zutreffend
libcufft.so.9.0	nicht zutreffend
libcurand.so.9.0	nicht zutreffend
libcusolver.so.9.0	nicht zutreffend
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libnvinfer.so.4	nicht zutreffend
libnvm_gpu.so	nicht zutreffend
libnvm.so	nicht zutreffend

Bibliothek	Mindestversion
libnvidia-fatbinaryloader.so.28.2.1	nicht zutreffend
libnvos.so	nicht zutreffend
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Object Detection x86_64

Bibliothek	Mindestversion
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

Bibliothek	Mindestversion
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0

Bibliothek	Mindestversion
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problembereiche

Symptom	Lösung
Auf einem Raspberry Pi wird die folgende Fehlermeldung protokolliert, und Sie verwenden nicht die Kamera: <code>Failed to initialize libdc1394</code>	Führen Sie den folgenden Befehl aus, um den Treiber zu deaktivieren: <pre>sudo ln /dev/null /dev/raw1394</pre> Diese Operation ist flüchtig. Der symbolische Link ist nach dem Neustart nicht mehr vorhanden. Schlagen Sie im Handbuch Ihrer Betriebssystemverteilung nach, wie der Link beim Neustart automatisch erstellt werden kann.

Lizenzen

Die ML Object Detection-Konnektoren beinhalten die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz

- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz

- [Deep Learning Runtime](#)/Apache-Lizenz 2.0
- [six](#)/MIT

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Durchführen von Machine Learning-Inferenzen](#)
- [Algorithmus zur Objekterkennung](#) im Amazon SageMaker AI Developer Guide

Modbus-RTU-Protokolladapter-Anschluss

Der [Modbus-RTU-Protokolladapter-Anschluss](#) fragt Informationen von Modbus RTU-Geräten ab, die sich in der Gruppe befinden. AWS IoT Greengrass

Dieser Konnektor empfängt Parameter für eine Modbus-RTU-Anforderung von einer benutzerdefinierten Lambda-Funktion. Er sendet die entsprechende Anforderung und veröffentlicht dann die Antwort des Zielgeräts als MQTT-Nachricht.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusRTUProtocolAdapter/versions/3

Version	ARN
2	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusRTUProtocolAdapter/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusRTUProtocolAdapter/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Eine physische Verbindung zwischen dem AWS IoT Greengrass Core und den Modbus-Geräten. Der Core muss physisch über eine serielle Schnittstelle (z. B. eine USB-Schnittstelle) mit dem Modbus RTU-Netzwerk verbunden sein.
- Eine [lokale Gerätere Ressource](#) in der Greengrass-Gruppe, die auf die physische serielle Modbus-Schnittstelle verweist.

- Eine benutzerdefinierte Lambda-Funktion, die Modbus RTU-Anforderungsparameter an diesen Konnektor sendet. Die Anforderungsparameter müssen den erwarteten Mustern entsprechen und die Adressen der IDs Zielgeräte im Modbus RTU-Netzwerk enthalten. Weitere Informationen finden Sie unter [the section called “Eingabedaten”](#).

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Eine physische Verbindung zwischen dem AWS IoT Greengrass Core und den Modbus-Geräten. Der Core muss physisch über eine serielle Schnittstelle (z. B. eine USB-Schnittstelle) mit dem Modbus RTU-Netzwerk verbunden sein.
- Eine [lokale Geräteressource](#) in der Greengrass-Gruppe, die auf die physische serielle Modbus-Schnittstelle verweist.
- Eine benutzerdefinierte Lambda-Funktion, die Modbus RTU-Anforderungsparameter an diesen Konnektor sendet. Die Anforderungsparameter müssen den erwarteten Mustern entsprechen und die Adressen der IDs Zielgeräte im Modbus RTU-Netzwerk enthalten. Weitere Informationen finden Sie unter [the section called “Eingabedaten”](#).

Steckerparameter

Dieser Konnektor unterstützt die folgenden Parameter:

ModbusSerialPort-ResourceId

Die ID der lokalen Geräteressource, die die physische serielle Modbus-Schnittstelle darstellt.

Note

Dieser Konnektor hat Zugriff zum Lesen und Schreiben auf die Ressource.

Anzeigenname in der AWS IoT Konsole: Ressource für die serielle Modbus-Port-Schnittstelle

Erforderlich: true

Typ: string

Gültiges Muster: .+

ModbusSerialPort

Der absolute Pfad zur physikalischen seriellen Modbus-Schnittstelle auf dem Gerät. Dies ist der Quellpfad, der für die lokale Modbus-Geräteressource angegeben ist.

Anzeigenname in der AWS IoT Konsole: Quellpfad der Ressource für die serielle Modbus-Port-Schnittstelle

Erforderlich: true

Typ: string

Gültiges Muster: .+

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt einen ConnectorDefinition mit einer ersten Version, die den Modbus-RTU-Protokolladapter-Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyModbusRTUProtocolAdapterConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
ModbusRTUProtocolAdapter/versions/3",  
      "Parameters": {  
        "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",  
        "ModbusSerialPort": "/path-to-port"  
      }  
    }  
  ]  
}'
```

Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie auf der Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

 Note

Nachdem Sie den Modbus-RTU-Protokolladapter-Connector bereitgestellt haben, können Sie ihn verwenden, AWS IoT Things Graph um Interaktionen zwischen Geräten in Ihrer Gruppe zu orchestrieren. Weitere Informationen finden Sie unter [Modbus](#) im AWS IoT Things Graph - Benutzerhandbuch.

Eingabedaten

Dieser Konnektor akzeptiert Modbus RTU-Anforderungsparameter von einer benutzerdefinierten Lambda-Funktion zu einem MQTT-Thema. Eingabenachrichten müssen im JSON-Format vorliegen.

Themenfilter im Abonnement

`modbus/adapter/request`

Nachrichten-Eigenschaften

Die Anforderungsnachricht variiert je nach Art der Modbus RTU-Anfrage, die sie darstellt. Die folgenden Eigenschaften werden für alle Anforderungen benötigt:

- Im `request`-Objekt:
 - `operation`. Der Name der auszuführenden Operation. Geben Sie beispielsweise an, dass `"operation": "ReadCoilsRequest"` Spulen lesen soll. Dieser Wert muss eine Unicode-Zeichenfolge sein. Informationen zu unterstützten Vorgängen finden Sie unter [the section called “Modbus RTU-Anforderungen und -Antworten”](#).
 - `device`. Das Zielgerät der Anfrage. Dieser Wert muss zwischen 0 - 247 liegen.
- Die `id`-Eigenschaft. Eine ID für die Anforderung. Dieser Wert wird für die Datenduplizierung verwendet und wie bei der `id`-Eigenschaft aller Antworten, einschließlich Fehlerantworten, zurückgegeben. Dieser Wert muss eine Unicode-Zeichenfolge sein.

 Note

Wenn Ihre Anforderung ein Adressfeld enthält, müssen Sie den Wert als Ganzzahl angeben. Beispiel, "address": 1.

Die weiteren Parameter, die in die Anforderung aufgenommen werden sollen, hängen von der Operation ab. Alle Anforderungsparameter sind erforderlich, mit Ausnahme der CRC, die separat behandelt wird. Beispiele finden Sie unter [the section called “Beispiel: Anforderungen und Antworten”](#).

Beispieleingabe: Lesen Spulenanforderungen

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Ausgabedaten

Dieser Konnektor veröffentlicht Antworten auf eingehende Modbus RTU-Anfragen.

Themenfilter im Abonnement

modbus/adapter/response

Nachrichten-Eigenschaften

Das Format der Antwortnachricht variiert je nach Anforderung und Antwortstatus. Beispiele finden Sie unter [the section called “Beispiel: Anforderungen und Antworten”](#).

 Note

Eine Antwort für einen Schreibvorgang ist lediglich ein Echo der Anforderung. Obwohl keine aussagekräftigen Informationen für Schreibantworten zurückgegeben werden, ist es eine gute Vorgehensweise, den Status der Antwort zu überprüfen.

Jede Antwort beinhaltet die folgenden Eigenschaften:

- Im response-Objekt:
 - `status`. Der Status der Anfrage. Der Status kann einer der folgenden Werte sein:
 - `Success`. Die Anfrage war gültig, wurde an das Modbus RTU-Netzwerk gesendet und eine Antwort wurde zurückgegeben.
 - `Exception`. Die Anfrage war gültig, wurde an das Modbus RTU-Netzwerk gesendet und es wurde eine Ausnahmeantwort zurückgegeben. Weitere Informationen finden Sie unter [the section called “Antwortstatus: Ausnahme”](#).
 - `No Response`. Die Anfrage war ungültig und der Connector hat den Fehler erkannt, bevor die Anfrage über das Modbus RTU-Netzwerk gesendet wurde. Weitere Informationen finden Sie unter [the section called “Antwortstatus: Keine Antwort”](#).
 - `device`. Das Gerät, an das die Anfrage gesendet wurde.
 - `operation`. Der Anfragetyp, der gesendet wurde.
 - `payload`. Der Inhalt der Antwort, der zurückgegeben wurde. Wenn der status `No Response` ist, enthält dieses Objekt nur eine `error`-Eigenschaft mit der Fehlerbeschreibung (z. B. `"error": "[Input/Output] No Response received from the remote unit"`).
- Die `id`-Eigenschaft. Die ID der Anforderung, die für die Datenduplizierung verwendet wird.

Beispielausgabe: Erfolg

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  }
}
```

```
    },  
    "id" : "TestRequest"  
  }  
}
```

Beispielausgabe: Fehler

```
{  
  "response" : {  
    "status" : "fail",  
    "error_message": "Internal Error",  
    "error": "Exception",  
    "device": 1,  
    "operation": "ReadCoilsRequest",  
    "payload": {  
      "function_code": 129,  
      "exception_code": 2  
    }  
  },  
  "id" : "TestRequest"  
}
```

Weitere Beispiele finden Sie unter [the section called “Beispiel: Anforderungen und Antworten”](#).

Modbus RTU-Anforderungen und -Antworten

Dieser Konnektor akzeptiert Modbus RTU-Anfrageparameter als [Eingabedaten](#) und veröffentlicht Antworten als [Ausgabedaten](#).

Die folgenden allgemeinen Operationen werden unterstützt.

Vorgangsname in Anforderung	Funktionscode in Antwort
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05

Vorgangsname in Anforderung	Funktionscode in Antwort
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

Beispiel: Anforderungen und Antworten

Im Folgenden finden Sie Beispiele für Anfragen und Antworten für unterstützte Operationen.

Spulen lesen

Anfragebeispiel:

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Antwortbeispiel:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  }
}
```

```
    },  
    "id" : "TestRequest"  
  }  
}
```

Digitaleingänge lesen

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "ReadDiscreteInputsRequest",  
    "device": 1,  
    "address": 1,  
    "count": 1  
  },  
  "id": "TestRequest"  
}
```

Antwortbeispiel:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "ReadDiscreteInputsRequest",  
    "payload": {  
      "function_code": 2,  
      "bits": [1]  
    }  
  },  
  "id" : "TestRequest"  
}
```

Lesen von Holdingregistern

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "ReadHoldingRegistersRequest",  
    "device": 1,  
    "address": 1,  
    "count": 1  
  },  
  "id": "TestRequest"  
}
```

```
  },  
  "id": "TestRequest"  
}
```

Antwortbeispiel:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "ReadHoldingRegistersRequest",  
    "payload": {  
      "function_code": 3,  
      "registers": [20,30]  
    }  
  },  
  "id" : "TestRequest"  
}
```

Lesen von Eingangsregistern

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "ReadInputRegistersRequest",  
    "device": 1,  
    "address": 1,  
    "value": 1  
  },  
  "id": "TestRequest"  
}
```

Schreiben Einzelspule

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "WriteSingleCoilRequest",  
    "device": 1,  
    "address": 1,  
    "value": 1  
  }
```

```
  },  
  "id": "TestRequest"  
}
```

Antwortbeispiel:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteSingleCoilRequest",  
    "payload": {  
      "function_code": 5,  
      "address": 1,  
      "value": true  
    }  
  },  
  "id" : "TestRequest"
```

Schreiben eines Einzelregisters

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "WriteSingleRegisterRequest",  
    "device": 1,  
    "address": 1,  
    "value": 1  
  },  
  "id": "TestRequest"  
}
```

Schreiben mehrerer Spulen

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "WriteMultipleCoilsRequest",  
    "device": 1,  
    "address": 1,  
    "values": [1,0,0,1]
```

```
  },  
  "id": "TestRequest"  
}
```

Antwortbeispiel:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteMultipleCoilsRequest",  
    "payload": {  
      "function_code": 15,  
      "address": 1,  
      "count": 4  
    }  
  },  
  "id" : "TestRequest"  
}
```

Schreiben mehrerer Register

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "WriteMultipleRegistersRequest",  
    "device": 1,  
    "address": 1,  
    "values": [20,30,10]  
  },  
  "id": "TestRequest"  
}
```

Antwortbeispiel:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "WriteMultipleRegistersRequest",  
    "payload": {  
      "function_code": 23,  

```



```
        "address": 1,  
        "count": 3  
    },  
    },  
    "id" : "TestRequest"  
}
```

Masken-Schreibregister

Anfragebeispiel:

```
{  
  "request": {  
    "operation": "MaskWriteRegisterRequest",  
    "device": 1,  
    "address": 1,  
    "and_mask": 175,  
    "or_mask": 1  
  },  
  "id": "TestRequest"  
}
```

Antwortbeispiel:

```
{  
  "response": {  
    "status": "success",  
    "device": 1,  
    "operation": "MaskWriteRegisterRequest",  
    "payload": {  
      "function_code": 22,  
      "and_mask": 0,  
      "or_mask": 8  
    }  
  },  
  "id" : "TestRequest"  
}
```

Lesen mehrerer Register

Anfragebeispiel:

```
{
```

```
"request": {
  "operation": "ReadWriteMultipleRegistersRequest",
  "device": 1,
  "read_address": 1,
  "read_count": 2,
  "write_address": 3,
  "write_registers": [20,30,40]
},
"id": "TestRequest"
}
```

Antwortbeispiel:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadWriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "registers": [10,20,10,20]
    }
  },
  "id" : "TestRequest"
}
```

Note

Die in dieser Antwort zurückgegebenen Register sind die Register, aus denen gelesen wird.

Antwortstatus: Ausnahme

Ausnahmen können auftreten, wenn das Anfrageformat gültig ist, die Anfrage aber nicht erfolgreich abgeschlossen wurde. In diesem Fall enthält die Antwort die folgenden Informationen:

- Der status wird auf Exception gesetzt.
- Der function_code entspricht dem Funktionscode der Anforderung + 128.
- Der exception_code enthält den Ausnahmecode. Weitere Informationen finden Sie unter Modbus-Ausnahmecodes.

Beispiel:

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "TestRequest"
}
```

Antwortstatus: Keine Antwort

Dieser Konnektor führt Validierungsprüfungen für die Modbus-Anforderung durch. So wird beispielsweise nach ungültigen Formaten und fehlenden Feldern gesucht. Wenn die Validierung fehlschlägt, sendet der Konnektor die Anforderung nicht. Stattdessen gibt er eine Antwort zurück, die die folgenden Informationen enthält:

- Der status wird auf No Response gesetzt.
- Der error enthält den Grund für den Fehler.
- Die error_message enthält die Fehlermeldung.

Beispiele:

```
{
  "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 <type 'int'>, got <type 'str'>"
    }
  }
}
```

```
},  
  "id" : "TestRequest"  
}
```

Wenn die Anforderung auf ein nicht vorhandenes Gerät abzielt oder wenn das Modbus RTU-Netzwerk nicht funktioniert, erhalten Sie möglicherweise eine `ModbusIOException`, die das No Response-Format verwendet.

```
{  
  "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"  
}
```

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.
2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner `greengrasssdk` auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder `"Pinned": true` in der CLI).
 - b. Fügen Sie die erforderliche lokale Gerätereource hinzu und gewähren Sie Lese-/Schreibzugriff auf die Lambda-Funktion.
 - c. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - d. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder `"Pinned": false` in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
```

```
import json

TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
    request = {
        "request": {
            "operation": "ReadCoilsRequest",
            "device": 1,
            "address": 1,
            "count": 1
        },
        "id": "TestRequest"
    }
    return request

def publish_basic_request():
    iot_client.publish(payload=json.dumps(create_read_coils_request()),
        topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
    return
```

Lizenzen

Der Modbus-RTU-Protokolladapter-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [pymodbus](#)/BSD
- [pyserial](#)/BSD

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Der Connector-ARN wurde zur AWS-Region Unterstützung aktualisiert. Verbesserte Fehlerprotokollierung.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

Modbus-TCP-Protokolladapter-Anschluss

Der [Modbus-TCP-Protokolladapter-Connector](#) sammelt Daten von lokalen Geräten über das ModbusTCP-Protokoll und veröffentlicht sie in den ausgewählten Streams. StreamManager

Sie können diesen Connector auch mit dem SiteWise IoT-Connector und Ihrem SiteWise IoT-Gateway verwenden. Ihr Gateway muss die Konfiguration für den Connector bereitstellen.

Weitere Informationen finden [Sie unter Konfiguration einer Modbus-TCP-Quelle](#) im SiteWise IoT-Benutzerhandbuch.

Note

Dieser Connector wird im Modus „[Keine Container-Isolation](#)“ ausgeführt, sodass Sie ihn für eine AWS IoT Greengrass Gruppe bereitstellen können, die in einem Docker-Container ausgeführt wird.

Dieser Connector hat die folgenden Versionen.

Version	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusTCPConnector/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusTCPConnector/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/ModbusTCPConnector/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 1 - 3

- AWS IoT Greengrass Kernsoftware v1.10.2 oder höher.
- Stream Manager ist für die AWS IoT Greengrass Gruppe aktiviert.
- Java 8 wurde auf dem Core-Gerät installiert und der PATH Umgebungsvariablen hinzugefügt.

 Note

Dieser Connector ist nur in den folgenden Regionen verfügbar:

- ap-southeast-1
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2
- cn-north-1

Konnektor-Parameter

Dieser Konnektor unterstützt die folgenden Parameter:

LocalStoragePath

Das Verzeichnis auf dem AWS IoT Greengrass Host, in das der SiteWise IoT-Connector persistente Daten schreiben kann. Das Standardverzeichnis ist `/var/sitewise`.

Anzeigenname in der AWS IoT Konsole: Lokaler Speicherpfad

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$|\/`.

MaximumBufferSize

Die maximale Größe in GB für die Nutzung von SiteWise IoT-Festplatten. Die Standardgröße ist 10 GB.

Anzeigenname in der AWS IoT Konsole: Maximale Festplattenpuffergröße

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\s*$| [0-9]+`

CapabilityConfiguration

Der Satz von Modbus-TCP-Collector-Konfigurationen, von denen der Connector Daten sammelt und zu denen er eine Verbindung herstellt.

Anzeigenname in der AWS IoT Konsole: CapabilityConfiguration

Erforderlich: false

Typ: Eine wohlgeformte JSON-Zeichenfolge, die den Satz der unterstützten Feedback-Konfigurationen definiert.

Das Folgende ist ein Beispiel für eineCapabilityConfiguration:

```
{
  "sources": [
    {
      "type": "ModBusTCPSource",
      "name": "SourceName1",
      "measurementDataStreamPrefix": "SourceName1_Prefix",
      "destination": {
        "type": "StreamManager",
        "streamName": "SiteWise_Stream_1",
        "streamBufferSize": 8
      },
      "endpoint": {
        "ipAddress": "127.0.0.1",
        "port": 8081,
        "unitId": 1
      },
      "propertyGroups": [
        {
          "name": "GroupName",
          "tagPathDefinitions": [
            {
              "type": "ModBusTCPAddress",
              "tag": "TT-001",
              "address": "30001",
              "size": 2,
              "srcDataType": "float",
              "transformation": "byteWordSwap",
```

```

        "dstDataType": "double"
      }
    ],
    "scanMode": {
      "type": "POLL",
      "rate": 100
    }
  }
]
}
]
}
}

```

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt einen ConnectorDefinition mit einer ersten Version, die den Modbus-TCP-Protokolladapter-Connector enthält.

```

aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
  "Connectors": [
    {
      "Id": "MyModbusTCPConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
      "Parameters": {
        "capability_configuration": "{\"version\":1,\"namespace\":
        \"iotsitewise:modbuscollector:1\", \"configuration\": {\"sources\": [{\"type
        \": \"ModBusTCPSource\", \"name\": \"SourceName1\", \"measurementDataStreamPrefix
        \": \"\", \"endpoint\": {\"ipAddress\": \"127.0.0.1\", \"port\": 8081, \"unitId\": 1},
        \"propertyGroups\": [{\"name\": \"PropertyGroupName\", \"tagPathDefinitions\": [{\"type
        \": \"ModBusTCPAddress\", \"tag\": \"TT-001\", \"address\": \"30001\", \"size\": 2,
        \"srcDataType\": \"hexdump\", \"transformation\": \"noSwap\", \"dstDataType\": \"string
        \"}], \"scanMode\": {\"rate\": 200, \"type\": \"POLL\"}}], \"destination\": {\"type\":
        \"StreamManager\", \"streamName\": \"SiteWise_Stream\", \"streamBufferSize\": 10},
        \"minimumInterRequestDuration\": 200}}]}\"
      }
    }
  ]
}'

```

 Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

Eingabedaten

Dieser Konnektor akzeptiert keine MQTT-Nachrichten als Eingabedaten.

Ausgabedaten

Dieser Konnektor veröffentlicht Daten für `StreamManager`. Sie müssen den Zielnachrichtenstream konfigurieren. Die Ausgabenachrichten haben die folgende Struktur:

```
{
  "alias": "string",
  "messages": [
    {
      "name": "string",
      "value": boolean|double|integer|string,
      "timestamp": number,
      "quality": "string"
    }
  ]
}
```

Lizenzen

Der Modbus-TCP-Protokolladapter-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [Digitaler Petri Modbus](#)

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen	Datum
3 (empfohlen)	Diese Version enthält Fehlerkorrekturen.	22. Dezember 2021
2	Unterstützung für ASCII- und ISO8859 kodierte UTF8 Quellzeichenfolgen hinzugefügt.	24. Mai 2021
1	Erstversion.	15. Dezember 2020

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

Raspberry Pi GPIO-Anschluss

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der Raspberry [Pi-GPIO-Anschluss](#) steuert die GPIO-Pins (Allzweckeingang/-ausgang) eines Raspberry Pi-Kerngeräts.

Dieser Konnektor fragt Eingangspins in einem bestimmten Intervall ab und veröffentlicht Zustandsänderungen an MQTT-Themen. Es akzeptiert auch Lese- und Schreib Anforderungen als

MQTT-Nachrichten von benutzerdefinierten Lambda-Funktionen. Schreibenanforderungen werden verwendet, um den Pin auf Hoch- oder Niederspannung zu setzen.

Der Konnektor stellt Parameter zur Verfügung, mit denen Sie Eingangs- und Ausgangspins definieren. Dieses Verhalten wird vor der Bereitstellung der Gruppe konfiguriert. Es kann während der Laufzeit nicht geändert werden.

- Eingangspins können zum Empfangen von Daten von Peripheriegeräten verwendet werden.
- Ausgangspins können zur Steuerung von Peripheriegeräten oder zum Senden von Daten an Peripheriegeräte verwendet werden.

Sie können diesen Konnektor für viele Szenarien verwenden, wie z. B.:

- Steuerung der grünen, gelben und roten LED-Leuchten für eine Ampel.
- Steuern eines Lüfters (mit einem elektrischen Relais verbunden) basierend auf Daten eines Feuchtigkeitssensors.
- Alarmierung der Mitarbeiter in einem Einzelhandelsgeschäft auf Knopfdruck.
- Verwendung eines intelligenten Lichtschalters zur Steuerung anderer IoT-Geräte.

 Note

Dieser Konnektor ist nicht geeignet für Anwendungen, die Echtzeitanforderungen haben. Ereignisse mit kurzer Dauer können verpasst werden.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
3	arn:aws:greengrass: <i>region</i> ::/connectors/RaspberryPiGPIO/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/RaspberryPiGPIO/versions/2

Version	ARN
1	arn:aws:greengrass: <i>region</i> ::/connectors/RaspberryPiGPIO/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Raspberry Pi 4 Modell B oder Raspberry Pi 3 Modell B/B+. Sie müssen die PIN-Reihenfolge Ihres Raspberry Pi kennen. Weitere Informationen finden Sie unter [the section called “GPIO-Pin-Sequenz”](#).
- Eine [lokale Gerätere Ressource](#) in der Greengrass-Gruppe, die /dev/gpiomem auf dem Raspberry Pi verweist. Wenn Sie die Ressource in der Konsole erstellen, müssen Sie die Option Betriebssystemgruppenberechtigungen automatisch hinzufügen der Linux-Gruppe auswählen, der die Ressource gehört. Stellen Sie die GroupOwnerSetting.AutoAddGroupOwner Eigenschaft in der API auf ein true.
- Das auf dem Raspberry Pi installierte [RPi.GPIO-Modul](#). In Raspbian ist dieses Modul standardmäßig installiert. Mit dem folgenden Befehl können Sie es neu installieren:

```
sudo pip install RPi.GPIO
```

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

- Raspberry Pi 4 Modell B oder Raspberry Pi 3 Modell B/B+. Sie müssen die PIN-Reihenfolge Ihres Raspberry Pi kennen. Weitere Informationen finden Sie unter [the section called “GPIO-Pin-Sequenz”](#).
- Eine [lokale Geräteressource](#) in der Greengrass-Gruppe, die `/dev/gpiomem` auf dem Raspberry Pi verweist. Wenn Sie die Ressource in der Konsole erstellen, müssen Sie die Option Betriebssystemgruppenberechtigungen automatisch hinzufügen der Linux-Gruppe auswählen, der die Ressource gehört. Stellen Sie die `GroupOwnerSetting.AutoAddGroupOwner` Eigenschaft in der API auf `true`.
- Das auf dem Raspberry Pi installierte [RPi.GPIO-Modul](#). In Raspbian ist dieses Modul standardmäßig installiert. Mit dem folgenden Befehl können Sie es neu installieren:

```
sudo pip install RPi.GPIO
```

GPIO-Pin-Sequenz

Der Raspberry Pi-GPIO-Anschluss referenziert GPIO-Pins anhand des Nummerierungsschemas des zugrunde liegenden System on Chip (SoC), nicht anhand der physikalischen Anordnung der GPIO-Pins. Die physische Reihenfolge der Pins kann je nach Raspberry Pi-Version variieren. Weitere Informationen finden Sie unter [GPIO](#) in der Raspberry Pi-Dokumentation.

Der Konnektor kann nicht prüfen, ob die von Ihnen konfigurierten Ein- und Ausgangspins korrekt auf die zugrunde liegende Hardware Ihres Raspberry Pi passen. Wenn die Pin-Konfiguration ungültig ist, gibt der Konnektor einen Laufzeitfehler zurück, wenn er versucht, auf dem Gerät zu starten. Um dieses Problem zu beheben, konfigurieren Sie den Konnektor neu und stellen Sie ihn dann erneut bereit.

Note

Stellen Sie sicher, dass die Peripheriegeräte für GPIO-Pins richtig verdrahtet sind, um Schäden an den Komponenten zu vermeiden.

Steckerparameter

Dieser Konnektor stellt die folgenden Parameter bereit:

InputGpios

Eine kommasetrennte Liste von GPIO-Pin-Nummern, die als Eingänge konfiguriert werden können. Optional können Sie U anfügen, um den Pull-up-Widerstand eines Pins einzustellen, oder D, um den Pull-down-Widerstand einzustellen. Beispiel: "5, 6U, 7D".

Anzeigename in der AWS IoT Konsole: GPIO-Pins eingeben

Erforderlich: `false`. Sie müssen Eingangspins, Ausgangspins oder beides angeben.

Typ: `string`

Gültiges Muster: `^[0-9]+[UD]?([0-9]+[UD]?)*$`

InputPollPeriod

Das Intervall (in Millisekunden) zwischen jedem Polling-Vorgang, das die Eingangs-GPIO-Pins auf Zustandsänderungen überprüft. Der minimale Wert beträgt 1.

Dieser Wert hängt von Ihrem Szenario und der Art der abgerufenen Geräte ab. Ein Wert von 50 sollte beispielsweise schnell genug sein, um einen Tastendruck zu erkennen.

Anzeigename in der AWS IoT Konsole: Geben Sie den GPIO-Abfragezeitraum ein

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^[1-9][0-9]*$`

OutputGpios

Eine kommasetrennte Liste von GPIO-Pin-Nummern, die als Ausgänge konfiguriert werden können. Optional kann H angefügt werden, um einen hohen Zustand (1) zu setzen, oder L, um einen niedrigen Zustand (0) einzustellen. Beispiel: "8H, 9, 27L".

Anzeigename in der AWS IoT Konsole: Ausgangs-GPIO-Pins

Erforderlich: `false`. Sie müssen Eingangspins, Ausgangspins oder beides angeben.

Typ: `string`

Gültiges Muster: `^[0-9]+[HL]?([0-9]+[HL]?)*$`

GpioMem-ResourceId

Die ID der lokalen Gerätere Ressource, die `/dev/gpiomem` repräsentiert.

 Note

Dieser Konnektor hat Zugriff zum Lesen und Schreiben auf die Ressource.

Anzeigename in der AWS IoT Konsole: `for /dev/gpiomemRessourcengerät`

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den Raspberry Pi GPIO-Anschluss enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MyRaspberryPiGPIOConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/  
versions/3",  
      "Parameters": {  
        "GpioMem-ResourceId": "my-gpio-resource",  
        "InputGpios": "5,6U,7D",  
        "InputPollPeriod": 50,  
        "OutputGpios": "8H,9,27L"  
      }  
    }  
  ]  
}'
```

 Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert Lese- oder Schreibanforderungen für GPIO-Pins zu zwei MQTT-Themen.

- Leseanforderungen zum Thema `gpio/+ /+ /read`.
- Schreibanforderungen zum Thema `gpio/+ /+ /write`.

Um zu diesen Themen zu veröffentlichen, ersetzen Sie jeweils die Platzhalter + durch den Namen der Core-Objekte und die Nummer des Zielpins. Zum Beispiel:

```
gpio/core-thing-name/gpio-number/read
```

Note

Derzeit müssen Sie beim Erstellen eines Abonnements, das den Raspberry Pi GPIO-Connector verwendet, einen Wert für mindestens einen der Plus-Platzhalter im Thema angeben.

Themenfilter: `gpio/+ /+ /read`

Verwenden Sie dieses Thema, um den Konnektor anzuweisen, den Zustand des GPIO-Pins zu lesen, der im Thema angegeben ist.

Der Konnektor veröffentlicht die Antwort zum entsprechenden Ausgabethema (z. B. `gpio/core-thing-name/gpio-number/state`).

Nachrichten-Eigenschaften

Keine. Nachrichten, die an dieses Thema gesendet werden, werden ignoriert.

Themenfilter: `gpio/+ /+ /write`

Verwenden Sie dieses Thema, um Schreibanfragen an einen GPIO-Pin zu senden. Dadurch wird der Konnektor angewiesen, den GPIO-Pin, der im Thema angegeben ist, auf eine Nieder- oder Hochspannung einzustellen.

- 0 setzt den Pin auf Niederspannung.
- 1 setzt den Pin auf Hochspannung.

Der Konnektor veröffentlicht die Antwort zum entsprechenden /state-Thema (z. B. gpio/*core-thing-name*/*gpio-number*/state).

Nachrichten-Eigenschaften

Der Wert 0 oder 1 als ganze Zahl oder Zeichenkette.

Beispieleingabe

0

Ausgabedaten

Dieser Konnektor veröffentlicht Daten zu zwei Themen:

- Hohe oder niedrige Zustandsänderungen im Thema gpio/+ /+ /state.
- Fehler im Thema gpio/+ /error.

Themenfilter: gpio/+ /+ /state

Verwenden Sie dieses Thema, um auf Zustandsänderungen an den Eingangspins und Antworten auf Leseanforderungen zu warten. Der Konnektor gibt die Zeichenkette "0" zurück, wenn sich der Pin in einem niedrigen Zustand befindet, oder "1", wenn er sich in einem hohen Zustand befindet.

Bei der Veröffentlichung zu diesem Thema ersetzt der Konnektor die Platzhalter + durch den Namen des Core-Objekts bzw. den Zielpin. Zum Beispiel:

gpio/*core-thing-name*/*gpio-number*/state

Note

Derzeit müssen Sie beim Erstellen eines Abonnements, das den Raspberry Pi-GPIO-Connector verwendet, einen Wert für mindestens einen der Plus-Platzhalter im Thema angeben.

Beispielausgabe

```
0
```

Themenfilter: gpio/+/error

Verwenden Sie dieses Thema, um nach Fehlern zu suchen. Der Konnektor veröffentlicht zu diesem Thema aufgrund einer ungültigen Anforderung, (z. B., wenn eine Zustandsänderung an einem Eingangspin angefordert wird).

Bei der Veröffentlichung zu diesem Thema ersetzt der Konnektor den Platzhalter + durch den Namen des Core-Objekts.

Beispielausgabe

```
{
  "topic": "gpio/my-core-thing/22/write",
  "error": "Invalid GPIO operation",
  "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations
are not permitted."
}
```

Beispiel für eine Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.
2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": true in der CLI).
 - b. Fügen Sie die erforderliche lokale Gerätereource hinzu und gewähren Sie Lese-/Schreibzugriff auf die Lambda-Funktion.
 - c. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - d. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": false in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor. In diesem Beispiel werden Leseanfragen für einen Satz von GPIO-Eingangspins gesendet. Es zeigt, wie Sie Themen mit dem Namen des Core-Objekts und der PIN-Nummer erstellen.

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIO = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIO:
        read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der Raspberry Pi GPIO; -Anschluss beinhaltet die folgende Software/Lizenzierung von Drittanbietern:

- [RPI.GPIO /MIT](#)

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Der Connector-ARN wurde zur AWS-Region Unterstützung aktualisiert.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [GPIO](#) in der Raspberry Pi-Dokumentation

Serieller Stream-Anschluss

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der Serial Stream [Connector](#) liest und schreibt an eine serielle Schnittstelle eines AWS IoT Greengrass Core-Geräts.

Dieser Konnektor unterstützt zwei Betriebsarten:

- Read-On-Demand. Empfängt Lese- und Schreibanforderungen zu MQTT-Themen und veröffentlicht die Antwort des Lesevorgangs oder den Status des Schreibvorgangs.
- Polling-Read. Liest in regelmäßigen Abständen von der seriellen Schnittstelle. Dieser Modus unterstützt auch Read-On-Demand Anfragen.

 Note

Leseanforderungen sind auf eine maximale Leselänge von 63994 Bytes begrenzt.
Schreibanforderungen sind auf eine maximale Datenlänge von 128000 Bytes beschränkt.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
3	arn:aws:greengrass: <i>region</i> ::/ connectors/SerialStream/ versions/3
2	arn:aws:greengrass: <i>region</i> ::/ connectors/SerialStream/ versions/2
1	arn:aws:greengrass: <i>region</i> ::/ connectors/SerialStream/ versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Eine [lokale Geräteressource](#) in der Greengrass-Gruppe, die auf die serielle Zielschnittstelle verweist.

Note

Bevor Sie diesen Konnektor bereitstellen, empfehlen wir, die serielle Schnittstelle einzurichten und zu überprüfen, ob sie lesbar und beschreibbar ist.

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Eine [lokale Geräteressource](#) in der Greengrass-Gruppe, die auf die serielle Zielschnittstelle verweist.

 Note

Bevor Sie diesen Konnektor bereitstellen, empfehlen wir, die serielle Schnittstelle einzurichten und zu überprüfen, ob sie lesbar und beschreibbar ist.

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

BaudRate

Die Baudrate der seriellen Verbindung.

Anzeigename in der AWS IoT Konsole: Baudrate

Erforderlich: true

Typ: string

Zulässige Werte: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400

Gültiges Muster: `^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

Timeout

Der Timeout (in Sekunden) für einen Lesevorgang.

Anzeigename in der AWS IoT Konsole: Timeout

Erforderlich: true

Typ: string

Zulässige Werte: 1 - 59

Gültiges Muster: `^([1-9]|[1-5][0-9])$`

SerialPort

Der absolute Pfad zur physikalischen seriellen Schnittstelle auf dem Gerät. Dies ist der Quellpfad, der für die lokale Gerätereource angegeben ist.

Anzeigename in der AWS IoT Konsole: Serielle Schnittstelle

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

Die ID der lokalen Gerätere Ressource, die die physische serielle Schnittstelle darstellt.

Note

Dieser Konnektor hat Zugriff zum Lesen und Schreiben auf die Ressource.

Anzeigename in der AWS IoT Konsole: Ressource für serielle Schnittstelle

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `[a-zA-Z0-9_-]+`

`PollingRead`

Legt den Lesemodus fest: Polling-Read oder. Read-On-Demand

- Für den Polling-Read-Modus geben Sie `true` an. In diesem Modus werden nur die Eigenschaften `PollingInterval`, `PollingReadType` und `PollingReadLength` benötigt.
- Geben Sie für Read-On-Demand den Modus an. `false` In diesem Modus werden die Werte für Typ und Länge in der Leseanforderung angegeben.

Anzeigename in der AWS IoT Konsole: Lesemodus

Erforderlich: `true`

Typ: `string`

Zulässige Werte: `true`, `false`

Gültiges Muster: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

PollingReadLength

Die Länge der zu lesenden Daten (in Bytes) bei jedem Polling-Lesevorgang. Dies gilt nur bei Verwendung des Polling-Read-Modus.

Anzeigename in der AWS IoT Konsole: Leselänge beim Abfragen

Erforderlich: `false`. Diese Eigenschaft ist erforderlich, wenn `PollingRead` `true` ist.

Typ: `string`

Gültiges Muster: `^([1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])$`

PollingReadInterval

Das Intervall (in Sekunden), in dem das Lesen des Pollings stattfindet. Dies gilt nur bei Verwendung des Polling-Read-Modus.

Anzeigename in der AWS IoT Konsole: Leseintervall für Abfragen

Erforderlich: `false`. Diese Eigenschaft ist erforderlich, wenn `PollingRead` `true` ist.

Typ: `string`

Gültige Werte: 1 — 999

Gültiges Muster: `^([1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

PollingReadType

Der Typ der Daten, die der Polling-Thread liest. Dies gilt nur bei Verwendung des Polling-Read-Modus.

Anzeigename in der AWS IoT Konsole: Lesetyp der Abfrage

Erforderlich: `false`. Diese Eigenschaft ist erforderlich, wenn `PollingRead` `true` ist.

Typ: `string`

Zulässige Werte: `ascii`, `hex`

Gültiges Muster: `^([Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

RtsCts

Gibt an, ob die RTS/CTS-Flusssteuerung aktiviert werden soll. Der Standardwert ist `false`. Weitere Informationen finden Sie unter [RTS, CTS und RTR](#).

Anzeigename in der AWS IoT Konsole: RTS/CTS Flow Control

Erforderlich: `false`

Typ: `string`

Zulässige Werte: `true`, `false`

Gültiges Muster: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

XonXoff

Gibt an, ob die Software-Flusssteuerung aktiviert werden soll. Der Standardwert ist `false`. Weitere Informationen finden Sie unter [Software-Flusssteuerung](#).

Anzeigename in der AWS IoT Konsole: Softwareflusssteuerung

Erforderlich: `false`

Typ: `string`

Zulässige Werte: `true`, `false`

Gültiges Muster: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

Parity

Die Parität der seriellen Schnittstelle. Der Standardwert ist `N`. Weitere Informationen finden Sie unter [Parität](#).

Anzeigename in der AWS IoT Konsole: Parität der seriellen Schnittstelle

Erforderlich: `false`

Typ: `string`

Zulässige Werte: `N`, `E`, `O`, `S`, `M`

Gültiges Muster: `^([NEOSMneosm])$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den Serial Stream-Connector enthält. Konfiguriert den Konnektor für den Polling-Read-Modus.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MySerialStreamConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/  
versions/3",  
      "Parameters": {  
        "BaudRate" : "9600",  
        "Timeout" : "25",  
        "SerialPort" : "/dev/serial1",  
        "SerialPort-ResourceId" : "my-serial-port-resource",  
        "PollingRead" : "true",  
        "PollingReadLength" : "30",  
        "PollingReadInterval" : "30",  
        "PollingReadType" : "hex"  
      }  
    }  
  ]  
}'
```

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert Lese- oder Schreibanforderungen für serielle Schnittstellen zu zwei MQTT-Themen. Eingabenachrichten müssen im JSON-Format sein.

- Leseanforderungen zum Thema `serial/+/read/#`.
- Schreibanforderungen zum Thema `serial/+/write/#`.

Um zu diesen Themen zu veröffentlichen, ersetzen Sie den Platzhalter `+` durch den Namen des Core-Objekts und den Platzhalter `#` durch den Pfad zur seriellen Schnittstelle. Zum Beispiel:

```
serial/core-thing-name/read/dev/serial-port
```

Themenfilter: serial/+ /read/#

Verwenden Sie dieses Thema, um On-Demand-Leseanforderungen an einen seriellen Pin zu senden. Leseanforderungen sind auf eine maximale Leselänge von 63994 Bytes begrenzt.

Nachrichten-Eigenschaften

`readLength`

Die Länge der Daten, die von der seriellen Schnittstelle gelesen werden sollen.

Erforderlich: true

Typ: string

Gültiges Muster: `^[1-9][0-9]*$`

`type`

Der Typ der zu lesenden Daten.

Erforderlich: true

Typ: string

Zulässige Werte: `ascii`, `hex`

Gültiges Muster: `(?i)^(ascii|hex)$`

`id`

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen.

Erforderlich: false

Typ: string

Gültiges Muster: `.+`

Beispieleingabe

```
{  
  "readLength": "30",  
}
```



```
"type": "ascii",  
"id": "abc123"  
}
```

Themenfilter: `serial/+/write/#`

Verwenden Sie dieses Thema, um Schreibanforderungen an einen seriellen Pin zu senden. Schreibanforderungen sind auf eine maximale Datenlänge von 128000 Bytes beschränkt.

Nachrichten-Eigenschaften

`data`

Die Zeichenfolge, die auf die serielle Schnittstelle geschrieben werden soll.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^[1-9][0-9]*$`

`type`

Der Typ der zu lesenden Daten.

Erforderlich: `true`

Typ: `string`

Zulässige Werte: `ascii, hex`

Gültiges Muster: `^(ascii|hex|ASCII|HEX)$`

`id`

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `.+`

Beispiel-Eingabe: ASCII-Anforderung

```
{
```

```
{
  "data": "random serial data",
  "type": "ascii",
  "id": "abc123"
}
```

Beispiel-Eingabe: Hex-Anforderung

```
{
  "data": "base64 encoded data",
  "type": "hex",
  "id": "abc123"
}
```

Ausgabedaten

Der Konnektor veröffentlicht Ausgabedaten zu zwei Themen:

- Statusinformationen vom Konnektor im Thema `serial/+/status/#`.
- Antworten auf Leseanforderungen zum Thema `serial/+/read_response/#`.

Bei der Veröffentlichung zu diesem Thema ersetzt der Konnektor den Platzhalter `+` durch den Namen des Core-Objekts und den Platzhalter `#` durch den Pfad zur seriellen Schnittstelle. Zum Beispiel:

```
serial/core-thing-name/status/dev/serial-port
```

Themenfilter: `serial/+/status/#`

Verwenden Sie dieses Thema, um den Status von Lese- und Schreibanforderungen anzuhören. Wenn eine `id`-Eigenschaft in die Anfrage aufgenommen wird, wird sie in der Antwort zurückgegeben.

Beispielausgabe: Erfolg

```
{
  "response": {
    "status": "success"
  },
  "id": "abc123"
}
```

Beispielausgabe: Fehler

Eine Fehlerreaktion beinhaltet eine `error_message`-Eigenschaft, die den Fehler oder das Timeout beschreibt, das beim Ausführen des Lese- oder Schreibvorgangs aufgetreten ist.

```
{
  "response": {
    "status": "fail",
    "error_message": "Could not write to port"
  },
  "id": "abc123"
}
```

Themenfilter: `serial/+/read_response/#`

Verwenden Sie dieses Thema, um Antwortdaten von einem Lesevorgang zu empfangen. Die Antwortdaten sind Base64-kodiert, wenn der Typ `hex` ist.

Beispielausgabe

```
{
  "data": "output of serial read operation"
  "id": "abc123"
}
```

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.

2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": true in der CLI).
 - b. Fügen Sie die erforderliche lokale Gerätereource hinzu und gewähren Sie Lese-/Schreibzugriff auf die Lambda-Funktion.
 - c. Fügen Sie Ihrer Gruppe den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - d. Fügen Sie der Gruppe Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": false in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
    request = {
        "data": "TEST",
        "type": "ascii",
        "id": "abc123"
    }
    return request

def publish_basic_request():
    iot_client.publish(payload=json.dumps(create_serial_stream_request()),
        topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
    return
```

Lizenzen

Der Serial Stream Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [pyserial](#)/BSD

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Der Connector-ARN wurde zur AWS-Region Unterstützung aktualisiert.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

ServiceNow MetricBase Integrationsanschluss

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bereitstellen. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der ServiceNow MetricBase [Integrationskonnektor](#) veröffentlicht Zeitreihenmetriken von Greengrass-Geräten an ServiceNow MetricBase. Auf diese Weise können Sie Zeitreihendaten aus der Greengrass Core-Umgebung speichern, analysieren und visualisieren und auf lokale Ereignisse reagieren.

Dieser Konnektor empfängt Zeitreihendaten zu einem MQTT-Thema und veröffentlicht die Daten in regelmäßigen Abständen in der ServiceNow API.

Sie können diesen Konnektor zur Unterstützung vieler Szenarien verwenden, darunter z. B.:

- Erstellen von Schwellenwert-basierten Warnmeldungen und Alarmen basierend auf Zeitreihendaten, die von Greengrass-Geräten gesammelt wurden.
- Verwenden Sie Zeitdienstdaten von Greengrass-Geräten mit benutzerdefinierten Anwendungen, die auf der ServiceNow Plattform erstellt wurden.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/2
1	arn:aws:greengrass: <i>region</i> ::/connectors/ServiceNowMetricBaseIntegration/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher. AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

 Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

 Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Ein ServiceNow Konto mit einem aktivierten Abonnement für MetricBase. Darüber hinaus muss im Konto eine Metrik und eine metrische Tabelle angelegt werden. Weitere Informationen finden Sie [MetricBase](#) in der ServiceNow Dokumentation.
- Ein geheimer Texttyp AWS Secrets Manager , in dem der Benutzername und das Passwort für die Anmeldung bei Ihrer ServiceNow Instance mit Standardauthentifizierung gespeichert werden. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten. Weitere Informationen finden Sie unter [Creating a Basic Secret](#) im AWS Secrets Manager Benutzerhandbuch.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Geheimnis verweist. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher. AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.



Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Ein ServiceNow Konto mit einem aktivierten Abonnement für MetricBase. Darüber hinaus muss im Konto eine Metrik und eine metrische Tabelle angelegt werden. Weitere Informationen finden Sie [MetricBase](#) in der ServiceNow Dokumentation.
- Ein geheimer Texttyp AWS Secrets Manager , in dem der Benutzername und das Passwort für die Anmeldung bei Ihrer ServiceNow Instance mit Standardauthentifizierung gespeichert werden. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten. Weitere Informationen finden Sie unter [Creating a Basic Secret](#) im AWS Secrets Manager Benutzerhandbuch.
- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Geheimnis verweist. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Version 4

PublishInterval

Die maximale Anzahl von Sekunden, bis zu der zwischen Veröffentlichungsereignissen gewartet werden soll ServiceNow. Der maximale Wert beträgt 900.

Der Connector veröffentlicht, bis ServiceNow er erreicht PublishBatchSize ist oder PublishInterval abläuft.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall in Sekunden

Erforderlich: true

Typ: string

Zulässige Werte: 1 - 900

Gültiges Muster: [1-9] | [1-9]\d | [1-9]\d\d | 900

PublishBatchSize

Die maximale Anzahl von Metrikwerten, die gebündelt werden können, bevor sie veröffentlicht werden. ServiceNow

Der Connector veröffentlicht bis zu dem ServiceNow Zeitpunkt, an dem er erreicht PublishBatchSize ist oder PublishInterval abläuft.

Anzeigename in der AWS IoT Konsole: Batchgröße veröffentlichen

Erforderlich: true

Typ: string

Gültiges Muster: ^[0-9]+\$

InstanceName

Der Name der Instanz, mit der eine Verbindung hergestellt wurde ServiceNow.

Anzeigename in der AWS IoT Konsole: Name der ServiceNow Instanz

Erforderlich: true

Typ: string

Gültiges Muster: .+

DefaultTableName

Der Name der Tabelle, die die mit der Zeitreihe `GlideRecord` verknüpfte `MetricBase` Datenbank enthält. Die Eigenschaft `table` in der Nutzlast der Input-Message kann verwendet werden, um diesen Wert zu überschreiben.

Anzeigename in der AWS IoT Konsole: Name der Tabelle, die die Metrik enthalten soll

Erforderlich: `true`

Typ: `string`

Gültiges Muster: .+

MaxMetricsToRetain

Die maximale Anzahl der Metriken, die im Speicher gespeichert werden können, bevor sie durch neue Metriken ersetzt werden.

Diese Begrenzung gilt, wenn keine Verbindung zum Internet besteht und der Konnektor beginnt, die Metriken zu puffern, um sie später zu veröffentlichen. Wenn der Puffer voll ist, werden die ältesten Metriken durch neue Metriken ersetzt.

 Note

Metriken werden nicht gespeichert, wenn der Host-Prozess für den Konnektor unterbrochen wird. Dies kann beispielsweise während der Gruppen-Bereitstellung oder beim Neustart des Geräts geschehen.

Dieser Wert sollte größer als die Batch-Größe und groß genug sein, um Nachrichten basierend auf der Eingangsrate der MQTT-Nachrichten zu halten.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl von Metriken, die im Speicher aufbewahrt werden sollen

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `^[0-9]+$`

AuthSecretArn

Das AWS Secrets Manager darin enthaltene Geheimnis speichert den ServiceNow Benutzernamen und das Passwort. Dies muss ein Texttyp Secret sein. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten.

Anzeigename in der AWS IoT Konsole: ARN of Auth Secret

Erforderlich: true

Typ: string

Gültiges Muster: `arn:aws:secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=, .@-]+-[a-zA-Z0-9]+`

AuthSecretArn-ResourceId

Die geheime Ressource in der Gruppe, die auf das Secrets Manager Manager-Geheimnis für die ServiceNow Anmeldeinformationen verweist.

Anzeigename in der AWS IoT Konsole: Auth-Token-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: `.+`

IsolationMode

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: false

Typ: string

Gültige Werte: GreengrassContainer oder NoContainer.

Gültiges Muster: ^NoContainer\$|^GreengrassContainer\$

Version 1 - 3

PublishInterval

Die maximale Anzahl von Sekunden, bis zu der zwischen Veröffentlichungsereignissen gewartet werden soll ServiceNow. Der maximale Wert beträgt 900.

Der Connector veröffentlicht, bis ServiceNow er erreicht PublishBatchSize ist oder PublishInterval abläuft.

Anzeigename in der AWS IoT Konsole: Veröffentlichungsintervall in Sekunden

Erforderlich: true

Typ: string

Zulässige Werte: 1 - 900

Gültiges Muster: [1-9]| [1-9]\d| [1-9]\d\d|900

PublishBatchSize

Die maximale Anzahl von Metrikwerten, die gebündelt werden können, bevor sie veröffentlicht werden. ServiceNow

Der Connector veröffentlicht bis zu dem ServiceNow Zeitpunkt, an dem er erreicht PublishBatchSize ist oder PublishInterval abläuft.

Anzeigename in der AWS IoT Konsole: Batchgröße veröffentlichen

Erforderlich: true

Typ: string

Gültiges Muster: ^[0-9]+\$

InstanceName

Der Name der Instanz, mit der eine Verbindung hergestellt wurde ServiceNow.

Anzeigename in der AWS IoT Konsole: Name der ServiceNow Instanz

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

`DefaultTableName`

Der Name der Tabelle, die die mit der Zeitreihe `GlideRecord` verknüpfte `MetricBase` Datenbank enthält. Die Eigenschaft `table` in der Nutzlast der Input-Message kann verwendet werden, um diesen Wert zu überschreiben.

Anzeigenname in der AWS IoT Konsole: Name der Tabelle, die die Metrik enthalten soll

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

`MaxMetricsToRetain`

Die maximale Anzahl der Metriken, die im Speicher gespeichert werden können, bevor sie durch neue Metriken ersetzt werden.

Diese Begrenzung gilt, wenn keine Verbindung zum Internet besteht und der Konnektor beginnt, die Metriken zu puffern, um sie später zu veröffentlichen. Wenn der Puffer voll ist, werden die ältesten Metriken durch neue Metriken ersetzt.

 Note

Metriken werden nicht gespeichert, wenn der Host-Prozess für den Konnektor unterbrochen wird. Dies kann beispielsweise während der Gruppen-Bereitstellung oder beim Neustart des Geräts geschehen.

Dieser Wert sollte größer als die Batch-Größe und groß genug sein, um Nachrichten basierend auf der Eingangsrate der MQTT-Nachrichten zu halten.

Anzeigenname in der AWS IoT Konsole: Maximale Anzahl von Metriken, die im Speicher aufbewahrt werden sollen

Erforderlich: `true`

Typ: string

Gültiges Muster: `^[0-9]+$`

AuthSecretArn

Das AWS Secrets Manager darin enthaltene Geheimnis speichert den ServiceNow Benutzernamen und das Passwort. Dies muss ein Texttyp Secret sein. Das Secret muss "Benutzer-" und "Passwort-"Schlüssel mit entsprechenden Werten enthalten.

Anzeigename in der AWS IoT Konsole: ARN of Auth Secret

Erforderlich: true

Typ: string

Gültiges Muster: `arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\-\+\/])*[a-zA-Z0-9/_+=,.\@-\+]-[a-zA-Z0-9]+`

AuthSecretArn-ResourceId

Die geheime Ressource in der Gruppe, die auf das Secrets Manager Manager-Geheimnis für die ServiceNow Anmeldeinformationen verweist.

Anzeigename in der AWS IoT Konsole: Auth-Token-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: `.+`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den ServiceNow MetricBase Integrationsconnector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyServiceNowMetricBaseIntegrationConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ServiceNowMetricBaseIntegration/versions/4",
```

```

    "Parameters": {
      "PublishInterval" : "10",
      "PublishBatchSize" : "50",
      "InstanceName" : "myinstance",
      "DefaultTableName" : "u_greengrass_app",
      "MaxMetricsToRetain" : "20000",
      "AuthSecretArn" : "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
      "AuthSecretArn-ResourceId" : "MySecretResource",
      "IsolationMode" : "GreengrassContainer"
    }
  ]
}'

```

Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert Zeitreihenmetriken zu einem MQTT-Thema und veröffentlicht die Metriken unter. ServiceNow Eingabenachrichten müssen im JSON-Format sein.

Themenfilter im Abonnement

```
servicenow/metricbase/metric
```

Nachrichten-Eigenschaften

```
request
```

Informationen über die Tabelle, den Datensatz und die Metrik. Diese Anforderung repräsentiert das Objekt `seriesRef` in einer Zeitreihen-POST-Anforderung. Weitere Informationen finden Sie unter [Clotho Zeitreihen-API - POST](#).

Erforderlich: `true`

Typ: object der die folgenden Eigenschaften beinhaltet:

subject

Die `sys_id` des spezifischen Datensatzes in der Tabelle.

Erforderlich: `true`

Typ: `string`

metric_name

Der Metrik-Feldname.

Erforderlich: `true`

Typ: `string`

table

Der Name der Tabelle, in der der Datensatz gespeichert werden soll. Geben Sie diesen Wert an, um den Parameter `DefaultTableName` zu überschreiben.

Erforderlich: `false`

Typ: `string`

value

Der Wert des einzelnen Datenpunktes.

Erforderlich: `true`

Typ: `float`

timestamp

Der Zeitstempel des einzelnen Datenpunktes. Der Standardwert ist die aktuelle Zeit.

Erforderlich: `false`

Typ: `string`

Beispieleingabe

```
{
  "request": {
    "subject": "ef43c6d40a0a0b5700c77f9bf387afe3",
    "metric_name": "u_count",
```

```
    "table": "u_greengrass_app"
    "value": 1.0,
    "timestamp": "2018-10-14T10:30:00"
  }
}
```

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema.

Themenfilter im Abonnement

`servicenow/metricbase/metric/status`

Beispielausgabe: Erfolg

```
{
  "response": {
    "metric_name": "Errors",
    "table_name": "GliderProd",
    "processed_on": "2018-10-14T10:35:00",
    "response_id": "khjKSkj132qwr23fcba",
    "status": "success",
    "values": [
      {
        "timestamp": "2016-10-14T10:30:00",
        "value": 1.0
      },
      {
        "timestamp": "2016-10-14T10:31:00",
        "value": 1.1
      }
    ]
  }
}
```

Beispielausgabe: Fehler

```
{
  "response": {
    "error": "InvalidInputException",
    "error_message": "metric value is invalid",
    "status": "fail"
  }
}
```

```
}  
}
```

Note

Wenn der Connector einen Fehler erkennt, der wiederholt werden kann (z. B. Verbindungsfehler), versucht er die Veröffentlichung im nächsten Batch erneut.

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.
2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": `true` in der CLI).

- b. Fügen Sie die erforderliche geheime Ressource hinzu und gewähren Sie der Lambda-Funktion Lesezugriff.
- c. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
- d. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": false in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
            "subject": '2efdf6badbd523803acfae441b961961',
            "metric_name": 'u_count',
            "value": 1234,
            "timestamp": '2018-10-20T20:22:20',
            "table": 'u_greengrass_metricbase_test'
        }
    }
```

```
}

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=SEND_TOPIC,
                       payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der ServiceNow MetricBase Integrationskonnektor umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [pysnow](#)/MIT

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
4	Der Parameter <code>IsolationMode</code> wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.

Version	Änderungen
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

SNS-Anschluss

Der [SNS-Connector](#) veröffentlicht Nachrichten zu einem Amazon SNS SNS-Thema. Auf diese Weise können Webserver, E-Mail-Adressen und andere Nachrichtenabonnenten auf Ereignisse in der Greengrass-Gruppe reagieren.

Dieser Konnektor empfängt SNS-Nachrichteninformationen zu einem MQTT-Thema und sendet die Nachricht dann an ein bestimmtes SNS-Thema. Sie können optional benutzerdefinierte Lambda-Funktionen verwenden, um Filter- oder Formatierungslogik für Nachrichten zu implementieren, bevor sie auf diesem Connector veröffentlicht werden.

Dieser Konnektor hat die folgenden Versionen.

Version	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/2

Version	ARN
1	arn:aws:greengrass: <i>region</i> ::/connectors/SNS/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher.
- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Ein konfiguriertes SNS-Thema. Weitere Informationen finden Sie unter [Amazon SNS-Thema anlegen](#) im Amazon Simple Notification Service-Entwicklerhandbuch.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `sns:Publish` Aktion auf dem Ziel-Arbeitsspeicher zulässt, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Sid": "Stmt1528133056761",
        "Action": [
            "sns:Publish"
        ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:sns:us-east-1:123456789012:topic-name"
        ]
    }
]
}

```

Mit diesem Konnektor können Sie das standardmäßige Thema in der Nutzlast der Input-Message dynamisch überschreiben. Wenn Ihre Implementierung diese Funktion verwendet, muss die IAM-Richtlinie `sns:Publish` Berechtigungen für alle Zielthemen zulassen. Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher.
- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Ein konfiguriertes SNS-Thema. Weitere Informationen finden Sie unter [Amazon SNS-Thema anlegen](#) im Amazon Simple Notification Service-Entwicklerhandbuch.
- Die [Greengrass-Gruppenrolle](#) ist so konfiguriert, dass sie die `sns:Publish` Aktion auf dem Ziel-Arizona zulässt `SNSTopic`, wie in der folgenden IAM-Beispielrichtlinie gezeigt.

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {

```



```
    "Sid": "Stmt1528133056761",
    "Action": [
        "sns:Publish"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:sns:us-east-1:123456789012:topic-name"
    ]
  }
}
```

Mit diesem Konnektor können Sie das standardmäßige Thema in der Nutzlast der Input-Message dynamisch überschreiben. Wenn Ihre Implementierung diese Funktion verwendet, muss die IAM-Richtlinie `sns:Publish` Berechtigungen für alle Zielthemen zulassen. Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren (etwa mit einem Benennungsschema mit Platzhaltern *).

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Version 4

DefaultSNSArn

Der ARN des Standard-SNS-Themas, in dem Nachrichten veröffentlicht werden sollen. Das Ziel-Thema kann durch die Eigenschaft `sns_topic_arn` in der Nutzlast der Eingangsnachricht überschrieben werden.

Note

Die Gruppenrolle muss die Berechtigung `sns:Publish` für alle Ziel-Themen erlauben. Weitere Informationen finden Sie unter [the section called “Voraussetzungen”](#).

Anzeigename in der AWS IoT Konsole: Standard-SNS-Thema ARN

Erforderlich: true

Typ: string

Gültiges Muster: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_\])+)`\$

IsolationMode

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

 Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: false

Typ: string

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültiges Muster: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

DefaultSNSArn

Der ARN des Standard-SNS-Themas, in dem Nachrichten veröffentlicht werden sollen. Das Ziel-Thema kann durch die Eigenschaft `sns_topic_arn` in der Nutzlast der Eingangsnachricht überschrieben werden.

 Note

Die Gruppenrolle muss die Berechtigung `sns:Publish` für alle Ziel-Themen erlauben. Weitere Informationen finden Sie unter [the section called "Voraussetzungen"](#).

Anzeigenname in der AWS IoT Konsole: Standard-SNS-Thema ARN

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `arn:aws:sns:([a-z]{2}-[a-z]+\d{1}):(\d{12}):([a-zA-Z0-9-_\d]{1,25})$`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den SNS-Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-  
version '{  
  "Connectors": [  
    {  
      "Id": "MySNSConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",  
      "Parameters": {  
        "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",  
        "IsolationMode" : "GreengrassContainer"  
      }  
    }  
  ]  
}'
```

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert SNS-Nachrichteninformationen zu einem MQTT-Thema und veröffentlicht die Nachricht dann unverändert im Ziel-SNS-Thema. Eingabenachrichten müssen im JSON-Format vorliegen.

Themenfilter im Abonnement

`sns/message`

Nachrichten-Eigenschaften

`request`

Informationen über die zu sendende Nachricht an das SNS-Thema.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`message`

Der Inhalt der Nachricht liegt als Zeichenfolge oder im JSON-Format vor. Beispiele finden Sie unter [Beispieleingabe](#).

Zum Senden von JSON muss die Eigenschaft `message_structure` auf `json` gesetzt sein, und die Nachricht muss eine zeichenfolgenkodierte JSON-Objekt mit einem `default`-Schlüssel sein.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.*`

`subject`

Der Betreff der Nachricht.

Erforderlich: `false`

Typ: ASCII-Text, bis zu 100 Zeichen. Dieser muss mit einem Buchstaben, einer Zahl oder einem Satzzeichen beginnen. Er darf keine Zeilenumbrüche oder Steuerzeichen enthalten.

Gültiges Muster: `.*`

`sns_topic_arn`

Der ARN des SNS-Themas, in dem Nachrichten veröffentlicht werden sollen. Wenn angegeben, veröffentlicht der Konnektor zu diesem Thema anstelle des Standardthemas.

 Note

Die Gruppenrolle muss die Berechtigung `sns:Publish` für jedes Ziel-Thema erlauben. Weitere Informationen finden Sie unter [the section called "Voraussetzungen"](#).

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_\+]*)$`

`message_structure`

Der Struktur der Nachricht.

Erforderlich: `false`. Dies muss angegeben werden, damit eine JSON-Nachricht gesendet werden kann.

Typ: `string`

Zulässige Werte: `json`

`id`

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen. Wenn angegeben, wird die Eigenschaft `id` im Antwortobjekt auf diesen Wert gesetzt. Wenn Sie diese Funktion nicht verwenden, können Sie diese Eigenschaft weglassen oder eine leere Zeichenkette angeben.

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `.*`

Beschränkungen

Die Nachrichtengröße ist durch eine maximale SNS-Nachrichtengröße von 256 KB begrenzt.

Beispieleingabe: Zeichenfolgen-Nachricht

Dieses Beispiel sendet eine Zeichenfolgen-Nachricht. Es legt die optionale `sns_topic_arn`-Eigenschaft fest, durch die das Standard-Ziel-Thema außer Kraft gesetzt wird.

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Beispieleingabe: JSON-Nachricht

Dieses Beispiel sendet eine Nachricht als zeichenfolgenkodierte JSON-Objekt mit dem `default`-Schlüssel.

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema.

Themenfilter im Abonnement

`sns/message/status`

Beispielausgabe: Erfolg

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
  }
}
```

```
    "status": "success"
  },
  "id": "request123"
}
```

Beispielausgabe: Fehler

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.

Für die Gruppenrollenanforderung müssen Sie die Rolle so konfigurieren, dass die erforderlichen Berechtigungen erteilt werden, und sicherstellen, dass die Rolle der Gruppe hinzugefügt wurde. Für weitere Informationen siehe [the section called “Verwalten der Gruppenrolle \(Konsole\)”](#) oder [the section called “Verwalten der Gruppenrolle \(CLI\)”](#).

2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": true in der CLI).
 - b. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - c. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": false in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
import time
import json
```



```
iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'

def create_request_with_all_fields():
    return {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der SNS-Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [AWS SDK für Python \(Boto3\)](#)/Apache-Lizenz 2.0
- [botocore](#)/Apache-Lizenz 2.0
- [dateutil](#)/PSF-Lizenz
- [docutils](#)/BSD-Lizenz, GNU General Public License (GPL), Python Software Foundation License, Public Domain
- [jmespath](#)/MIT-Lizenz
- [s3transfer](#)/Apache-Lizenz 2.0
- [urllib3](#)/MIT-Lizenz

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
4	Der Parameter <code>IsolationMode</code> wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Die Veröffentlichen-Aktion](#) in der Boto 3-Dokumentation
- [Was ist Amazon Simple Notification Service?](#) im Amazon Simple Notification Service-Entwicklerhandbuch

Konnektor für Splunk-Integration

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der Splunk Integration [Connector](#) veröffentlicht Daten von Greengrass-Geräten an Splunk. Dies ermöglicht es Ihnen, Splunk zur Überwachung und Analyse der Greengrass Core-Umgebung zu verwenden und auf lokale Ereignisse zu reagieren. Der Konnektor ist in den HTTP Event Collector (HEC) integriert. Weitere Informationen finden Sie unter [Einführung in den Splunk HTTP Event Collector](#) in der Splunk-Dokumentation.

Dieser Konnektor empfängt Protokoll- und Ereignisdaten zu einem MQTT-Thema und veröffentlicht die Daten an die Splunk-API.

Sie können diesen Konnektor zur Unterstützung von industriellen Szenarien verwenden, darunter z. B.:

- Bediener können periodische Daten von Aktuatoren und Sensoren (z. B. Temperatur-, Druck- und Wasserwerte) verwenden, um Alarme auszulösen, wenn Werte bestimmte Schwellenwerte überschreiten.
- Developer können Daten verwenden, die von Industriemaschinen gesammelt wurden, um ML-Modelle zu entwerfen, die die Anlagen auf mögliche Probleme überwachen.

Dieser Anschluss hat die folgenden Versionen.

Version	ARN
4	arn:aws:greengrass: <i>region</i> ::/connectors/SplunkIntegration/versions/4

Version	ARN
3	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/3</code>
2	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/2</code>
1	<code>arn:aws:greengrass: <i>region</i>::/connectors/SplunkIntegration/versions/1</code>

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 3 - 4

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher. AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

 Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Die Funktionalität des HTTP Event Collectors muss im Splunk aktiviert sein. Weitere Informationen finden Sie unter [Set up and use HTTP Event Collector in Splunk Web](#) in der Splunk-Dokumentation.
- Ein geheimer Texttyp AWS Secrets Manager, in dem Ihr Splunk HTTP Event Collector-Token gespeichert wird. Weitere Informationen finden Sie in der Splunk-Dokumentation unter [Über Event-Collector-Token](#) und unter [Creating a Basic Secret im Benutzerhandbuch](#).AWS Secrets Manager

 Note

Um das Geheimnis in der Secrets Manager-Konsole zu erstellen, geben Sie Ihr Token auf der Registerkarte Klartext ein. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für die `SecretString` Eigenschaft an.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Geheimnis verweist. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Versions 1 - 2

- AWS IoT Greengrass Kernsoftware v1.7 oder höher. AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

 Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Die Funktionalität des HTTP Event Collectors muss im Splunk aktiviert sein. Weitere Informationen finden Sie unter [Set up and use HTTP Event Collector in Splunk Web](#) in der Splunk-Dokumentation.
- Ein geheimer Texttyp AWS Secrets Manager , in dem Ihr Splunk HTTP Event Collector-Token gespeichert wird. Weitere Informationen finden Sie in der Splunk-Dokumentation unter [Über Event-Collector-Token](#) und unter [Creating a Basic Secret im Benutzerhandbuch](#).AWS Secrets Manager

 Note

Um das Geheimnis in der Secrets Manager-Konsole zu erstellen, geben Sie Ihr Token auf der Registerkarte Klartext ein. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für die SecretString Eigenschaft an.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Geheimnis verweist. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit:

Version 4

SplunkEndpoint

Der Endpunkt Ihrer Splunk-Instance. Dieser Wert muss das Protokoll, den Hostnamen und den Port enthalten.

Anzeigename in der AWS IoT Konsole: Splunk-Endpunkt

Erforderlich: true

Typ: string

Gültiges Muster: `^(http:\\\\|https:\\\\)?[a-z0-9]+([-\\.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\\/.*)?$`

MemorySize

Die Speichergröße (in KB), die dem Konnektor zugewiesen werden soll.

Anzeigename in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

SplunkQueueSize

Die maximale Anzahl von Elementen, die im Speicher gespeichert werden können, bevor die Elemente übertragen oder verworfen werden. Wenn dieses Limit erreicht wird, werden die ältesten Elemente in der Warteschlange durch neuere Elemente ersetzt. Diese Begrenzung kommt typischerweise zum tragen, wenn keine Verbindung zum Internet besteht.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl der beizubehaltenden Elemente

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

SplunkFlushIntervalSeconds

Das Intervall (in Sekunden) für die Veröffentlichung empfangener Daten an Splunk HEC. Der maximale Wert beträgt 900. Um den Konnektor so zu konfigurieren, dass er Elemente veröffentlicht, sobald sie empfangen werden (ohne Batching), geben Sie 0 an.

Anzeigename in der AWS IoT Konsole: Splunk-Veröffentlichungsintervall

Erforderlich: true

Typ: string

Gültiges Muster: `[0-9] | [1-9]\d | [1-9]\d\d | 900`

SplunkTokenSecretArn

Das Geheimnis AWS Secrets Manager , in dem das Splunk-Token gespeichert ist. Dies muss ein Texttyp Secret sein.

Anzeigename in der AWS IoT Konsole: ARN des geheimen Splunk-Authentifizierungstokens

Erforderlich: true

Typ: string

Gültiges Muster: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_\d]+-[a-zA-Z0-9-_\d]+`

SplunkTokenSecretArn-ResourceId

Die geheime Ressource in der Greengrass-Gruppe, die auf das Splunk-Secret verweist.

Anzeigename in der AWS IoT Konsole: Splunk-Authentifizierungstoken-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: `.+`

SplunkCustomCALocation

Der Dateipfad der benutzerdefinierten Zertifizierungsstelle (CA) für Splunk (z. B. `/etc/ssl/certs/splunk.crt`).

Anzeigename in der AWS IoT Konsole: Standort der benutzerdefinierten Splunk-Zertifizierungsstelle

Erforderlich: false

Typ: string

Gültiges Muster: ^\$|/.*

IsolationMode

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

 Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: false

Typ: string

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültiges Muster: ^NoContainer\$|^GreengrassContainer\$

Version 1 - 3

SplunkEndpoint

Der Endpunkt Ihrer Splunk-Instance. Dieser Wert muss das Protokoll, den Hostnamen und den Port enthalten.

Anzeigename in der AWS IoT Konsole: Splunk-Endpunkt

Erforderlich: true

Typ: string

Gültiges Muster: ^(http:\\\\|https:\\\\)?[a-z0-9]+([-\\.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\\/.*)?\$

MemorySize

Die Speichergröße (in KB), die dem Konnektor zugewiesen werden soll.

Anzeigename in der AWS IoT Konsole: Speichergröße

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

SplunkQueueSize

Die maximale Anzahl von Elementen, die im Speicher gespeichert werden können, bevor die Elemente übertragen oder verworfen werden. Wenn dieses Limit erreicht wird, werden die ältesten Elemente in der Warteschlange durch neuere Elemente ersetzt. Diese Begrenzung kommt typischerweise zum tragen, wenn keine Verbindung zum Internet besteht.

Anzeigename in der AWS IoT Konsole: Maximale Anzahl der beizubehaltenden Elemente

Erforderlich: true

Typ: string

Gültiges Muster: `^[0-9]+$`

SplunkFlushIntervalSeconds

Das Intervall (in Sekunden) für die Veröffentlichung empfangener Daten an Splunk HEC. Der maximale Wert beträgt 900. Um den Konnektor so zu konfigurieren, dass er Elemente veröffentlicht, sobald sie empfangen werden (ohne Batching), geben Sie 0 an.

Anzeigename in der AWS IoT Konsole: Splunk-Veröffentlichungsintervall

Erforderlich: true

Typ: string

Gültiges Muster: `[0-9] | [1-9]\d | [1-9]\d\d | 900`

SplunkTokenSecretArn

Das Geheimnis AWS Secrets Manager , in dem das Splunk-Token gespeichert ist. Dies muss ein Texttyp Secret sein.

Anzeigename in der AWS IoT Konsole: ARN des geheimen Splunk-Authentifizierungstokens

Erforderlich: true

Typ: string

Gültiges Muster: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_\d]{1,64}`

`SplunkTokenSecretArn-ResourceId`

Die geheime Ressource in der Greengrass-Gruppe, die auf das Splunk-Secret verweist.

Anzeigename in der AWS IoT Konsole: Splunk-Authentifizierungstoken-Ressource

Erforderlich: true

Typ: string

Gültiges Muster: `.+`

`SplunkCustomCALocation`

Der Dateipfad der benutzerdefinierten Zertifizierungsstelle (CA) für Splunk (z. B. `/etc/ssl/certs/splunk.crt`).

Anzeigename in der AWS IoT Konsole: Standort der benutzerdefinierten Splunk-Zertifizierungsstelle

Erforderlich: false

Typ: string

Gültiges Muster: `^$|/.*`

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Befehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den Splunk Integration Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MySplunkIntegrationConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/
versions/4",
```

```

    "Parameters": {
      "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",
      "MemorySize": 200000,
      "SplunkQueueSize": 10000,
      "SplunkFlushIntervalSeconds": 5,
      "SplunkTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
      "SplunkTokenSecretArn-ResourceId": "MySplunkResource",
      "IsolationMode" : "GreengrassContainer"
    }
  }
]
}'

```

Note

Die Lambda-Funktion in diesem Konnektor hat einen [langlebigen Lebenszyklus](#).

In der AWS IoT Greengrass Konsole können Sie über die Connectors-Seite der Gruppe einen Connector hinzufügen. Weitere Informationen finden Sie unter [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#).

Eingabedaten

Dieser Connector akzeptiert Protokollierungs- und Ereignisdaten zu einem MQTT-Thema und veröffentlicht die empfangenen Daten unverändert in der Splunk-API. Eingabenachrichten müssen im JSON-Format vorliegen.

Themenfilter im Abonnement

```
splunk/logs/put
```

Nachrichten-Eigenschaften

```
request
```

Die Ereignisdaten, die an die Splunk-API gesendet werden sollen. Ereignisse müssen den Spezifikationen der [Services/Collector](#)-API entsprechen.

Erforderlich: true

Typ:object. Nur die event-Eigenschaft ist erforderlich.

id

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung auf einen Ausgangsstatus abzubilden.

Erforderlich: false

Typ: string

Beschränkungen

Alle Beschränkungen, die durch die Splunk-API auferlegt werden, gelten für diesen Konnektor. Weitere Informationen finden Sie unter [Services/Collector](#).

Beispieleingabe

```
{
  "request": {
    "event": "some event",
    "fields": {
      "severity": "INFO",
      "category": [
        "value1",
        "value2"
      ]
    }
  },
  "id": "request123"
}
```

Ausgabedaten

Dieser Konnektor veröffentlicht Ausgabedaten zu zwei Themen:

- Statusinformationen zum Thema `splunk/logs/put/status`.
- Fehler im Thema `splunk/logs/put/error`.

Themenfilter: `splunk/logs/put/status`

Verwenden Sie dieses Thema, um den Status der Anforderungen anzuhören. Jedes Mal, wenn der Connector einen Stapel empfangener Daten an die Splunk-API sendet, veröffentlicht er eine Liste IDs der erfolgreichen und fehlgeschlagenen Anfragen.

Beispielausgabe

```
{
  "response": {
    "succeeded": [
      "request123",
      ...
    ],
    "failed": [
      "request789",
      ...
    ]
  }
}
```

Themenfilter: `splunk/logs/put/error`

Verwenden Sie dieses Thema, um nach Fehlern vom Konnektor zu suchen. Die Eigenschaft `error_message`, die den Fehler oder das Timeout beschreibt, die beim Verarbeiten der Anforderung aufgetreten sind.

Beispielausgabe

```
{
  "response": {
    "error": "UnauthorizedException",
    "error_message": "invalid splunk token",
    "status": "fail"
  }
}
```

Note

Wenn der Connector einen Fehler erkennt, der wiederholt werden kann (z. B. Verbindungsfehler), versucht er die Veröffentlichung im nächsten Batch erneut.

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

 Note

- Wenn Sie andere Python-Laufzeiten verwenden, können Sie einen Symlink von Python3.x zu Python 3.7 erstellen.
- In den Themen [Beginnen Sie mit Konnektoren \(Konsole\)](#) und [Erste Schritte mit Konnektoren \(CLI\)](#) wird ausführlich beschrieben, wie Sie einen Beispielkonnektor für Twilio-Benachrichtigungen konfigurieren und bereitstellen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.
2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.
 - a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": `true` in der CLI).
 - b. Fügen Sie die erforderliche geheime Ressource hinzu und gewähren Sie der Lambda-Funktion Lesezugriff.
 - c. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - d. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.

5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": `false` in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
4	Der Parameter <code>IsolationMode</code> wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.
3	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
2	Beheben, um übermäßige Protokollierung zu reduzieren.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

Konnektor für Twilio-Benachrichtigungen

Warning

Dieser Connector befindet sich in der Phase mit verlängerter Lebensdauer und veröffentlicht AWS IoT Greengrass keine Updates, die Funktionen, Verbesserungen vorhandener

Funktionen, Sicherheitspatches oder Bugfixes bieten. Weitere Informationen finden Sie unter [AWS IoT Greengrass Version 1 Wartungspolitik](#).

Der Twilio Notifications [Connector](#) tätigt automatisierte Telefonanrufe oder sendet Textnachrichten über Twilio. Mit diesem Konnektor können Sie Benachrichtigungen als Reaktion auf Ereignisse in der Greengrass-Gruppe senden. Bei Telefonaten kann der Konnektor eine Sprachnachricht an den Empfänger weiterleiten.

Dieser Konnektor empfängt Twilio-Nachrichteninformationen zu einem MQTT-Thema und löst dann eine Twilio-Benachrichtigung aus.

 Note

Eine Anleitung zur Verwendung des Twilio Notifications Connectors finden Sie unter oder [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#) [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)

Dieser Connector hat die folgenden Versionen.

Version	ARN
5	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/5
4	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/4
3	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/3
2	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/2

Version	ARN
1	arn:aws:greengrass: <i>region</i> ::/connectors/TwilioNotifications/versions/1

Informationen über Versionsänderungen finden Sie im [Änderungsprotokoll](#).

Voraussetzungen

Dieser Konnektor hat die folgenden Anforderungen:

Version 4 - 5

- AWS IoT Greengrass Kernsoftware v1.9.3 oder höher. AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- [Python-Version](#) 3.7 oder 3.8 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.

Note

Um Python 3.8 zu verwenden, führen Sie den folgenden Befehl aus, um einen symbolischen Link vom standardmäßigen Python 3.7-Installationsordner zu den installierten Python 3.8-Binärdateien zu erstellen.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt.

- Eine Twilio-Konto-SID, ein Auth-Token und eine Twilio-aktivierte Telefonnummer. Nachdem Sie ein Twilio-Projekt erstellt haben, stehen diese Werte im Projekt-Dashboard zur Verfügung.

 Note

Sie können ein Twilio-Testkonto verwenden. Wenn Sie ein Testkonto verwenden, müssen Sie einer Liste verifizierter Telefonnummern hinzufügen, die keine Twilio-Empfänger sind. Weitere Informationen finden Sie unter [So arbeiten Sie mit Ihrem kostenlosen Twilio-Testkonto](#).

- Ein geheimer Texttyp AWS Secrets Manager, in dem das Twilio-Authentifizierungstoken gespeichert wird. Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter [Erstellen eines grundlegenden Geheimnisses](#).

 Note

Um das Geheimnis in der Secrets Manager-Konsole zu erstellen, geben Sie Ihr Token auf der Registerkarte Klartext ein. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für die `SecretString` Eigenschaft an.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Geheimnis verweist. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Versions 1 - 3

- AWS IoT Greengrass Kernsoftware v1.7 oder höher. AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

 Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle

verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- [Python-Version](#) 2.7 wurde auf dem Core-Gerät installiert und zur Umgebungsvariablen PATH hinzugefügt.
- Eine Twilio-Konto-SID, ein Authent-Token und eine Twilio-aktivierte Telefonnummer. Nachdem Sie ein Twilio-Projekt erstellt haben, stehen diese Werte im Projekt-Dashboard zur Verfügung.

 Note

Sie können ein Twilio-Testkonto verwenden. Wenn Sie ein Testkonto verwenden, müssen Sie einer Liste verifizierter Telefonnummern hinzufügen, die keine Twilio-Empfänger sind. Weitere Informationen finden Sie unter [So arbeiten Sie mit Ihrem kostenlosen Twilio-Testkonto](#).

- Ein geheimer Texttyp AWS Secrets Manager , in dem das Twilio-Authentifizierungstoken gespeichert wird. Weitere Informationen finden Sie im AWS Secrets Manager Benutzerhandbuch unter [Erstellen eines grundlegenden Geheimnisses](#).

 Note

Um das Geheimnis in der Secrets Manager-Konsole zu erstellen, geben Sie Ihr Token auf der Registerkarte Klartext ein. Verwenden Sie keine Anführungszeichen oder andere Formatierungen. Geben Sie in der API das Token als Wert für die SecretString Eigenschaft an.

- Eine geheime Ressource in der Greengrass-Gruppe, die auf das Secrets Manager Manager-Geheimnis verweist. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Konnektor-Parameter

Dieser Konnektor stellt die folgenden Parameter bereit.

Version 5

TWILIO_ACCOUNT_SID

Die SID des Twilio-Kontos, mit der die Twilio-API aufgerufen wird.

Anzeigename in der AWS IoT Konsole: Twilio-Konto-SID

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

TwilioAuthTokenSecretArn

Der ARN des Secrets Manager, in dem das Twilio-Authentifizierungstoken gespeichert ist.

 Note

Dies wird verwendet, um auf den Wert des lokalen Secrets auf dem Core zuzugreifen.

Anzeigename in der AWS IoT Konsole: ARN des geheimen Twilio-Authentifizierungstokens

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.\@\\-]+-[a-zA-Z0-9]+`

TwilioAuthTokenSecretArn-ResourceId

Die ID der geheimen Ressource in der Greengrass-Gruppe, die auf das Secret für das Twilio-Authent-Token verweist.

Anzeigename in der AWS IoT Konsole: Twilio-Authentifizierungstoken-Ressource

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

DefaultFromPhoneNumber

Die standardmäßige Twilio-aktivierte Telefonnummer, die Twilio zum Senden von Nachrichten verwendet. Twilio verwendet diese Nummer, um die Textnachricht oder den Anruf zu initiieren.

- Wenn Sie keine Standard-Rufnummer konfigurieren, müssen Sie eine Rufnummer in der Eigenschaft `from_number` im Textkörper der Eingabemeldung angeben.
- Wenn Sie eine Standard-Rufnummer konfigurieren, können Sie die Standardeinstellung optional überschreiben, indem Sie die Eigenschaft `from_number` im Textkörper der Eingabemeldung angeben.

Anzeigename in der AWS IoT Konsole: Standardeinstellung aus der Telefonnummer

Erforderlich: `false`

Typ: `string`

Gültiges Muster: `^\$|\+[0-9]+`

IsolationMode

Der [Containerisierungsmodus](#) für diesen Konnektor. Die Standardeinstellung ist `GreengrassContainer`, was bedeutet, dass der Konnektor in einer isolierten Laufzeitumgebung innerhalb des AWS IoT Greengrass Containers ausgeführt wird.

Note

Die Standardeinstellung für Containerisierung für die Gruppe gilt nicht für Konnektoren.

Anzeigename in der AWS IoT Konsole: Container-Isolationsmodus

Erforderlich: `false`

Typ: `string`

Gültige Werte: `GreengrassContainer` oder `NoContainer`.

Gültiges Muster: `^NoContainer$|^GreengrassContainer$`

Version 1 - 4

TWILIO_ACCOUNT_SID

Die SID des Twilio-Kontos, mit der die Twilio-API aufgerufen wird.

Anzeigename in der AWS IoT Konsole: Twilio-Konto-SID

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

TwilioAuthTokenSecretArn

Der ARN des Secrets Manager, in dem das Twilio-Authentifizierungstoken gespeichert ist.

 Note

Dies wird verwendet, um auf den Wert des lokalen Secrets auf dem Core zuzugreifen.

Anzeigename in der AWS IoT Konsole: ARN des geheimen Twilio-Authentifizierungstokens

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `arn:aws:secretsmanager:[a-z0-9\-\+]:[0-9]{12}:secret:([a-zA-Z0-9\-\+\/]*[a-zA-Z0-9/_+=,.\@-\-]+-[a-zA-Z0-9]+)`

TwilioAuthTokenSecretArn-ResourceId

Die ID der geheimen Ressource in der Greengrass-Gruppe, die auf das Secret für das Twilio-Authent-Token verweist.

Anzeigename in der AWS IoT Konsole: Twilio-Authentifizierungstoken-Ressource

Erforderlich: `true`

Typ: `string`

Gültiges Muster: .+

DefaultFromPhoneNumber

Die standardmäßige Twilio-aktivierte Telefonnummer, die Twilio zum Senden von Nachrichten verwendet. Twilio verwendet diese Nummer, um die Textnachricht oder den Anruf zu initiieren.

- Wenn Sie keine Standard-Rufnummer konfigurieren, müssen Sie eine Rufnummer in der Eigenschaft `from_number` im Textkörper der Eingabemeldung angeben.
- Wenn Sie eine Standard-Rufnummer konfigurieren, können Sie die Standardeinstellung optional überschreiben, indem Sie die Eigenschaft `from_number` im Textkörper der Eingabemeldung angeben.

Anzeigename in der AWS IoT Konsole: Standardeinstellung aus der Telefonnummer

Erforderlich: false

Typ: string

Gültiges Muster: ^\$|\+[0-9]+

Beispiel für das Erstellen eines Konnektors (AWS CLI)

Der folgende CLI-Beispielbefehl erstellt eine `ConnectorDefinition` mit einer ersten Version, die den Twilio Notifications Connector enthält.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
  "Connectors": [
    {
      "Id": "MyTwilioNotificationsConnector",
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/5",
      "Parameters": {
        "TWILIO_ACCOUNT_SID": "abcd12345xyz",
        "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
        "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
        "DefaultFromPhoneNumber": "+19999999999",
        "IsolationMode" : "GreengrassContainer"
      }
    }
  ]
}
```

```
}'
```

Tutorials, die zeigen, wie der Twilio Notifications Connector zu einer Gruppe hinzugefügt wird, finden Sie unter [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#) und [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)

Eingabedaten

Dieser Connector akzeptiert Twilio-Nachrichteninformationen zu zwei MQTT-Themen. Eingabenachrichten müssen im JSON-Format vorliegen.

- Textnachrichteninformationen zum Thema `twilio/txt`.
- Telefonnachrichteninformationen zum Thema `twilio/call`.

Note

Die Nutzlast der Eingabemeldung kann eine Textnachricht (`message`) oder eine Sprachnachricht (`voice_message_location`) beinhalten, aber nicht beide.

Themenfilter: **twilio/txt**

Nachrichten-Eigenschaften

`request`

Informationen über die Twilio-Benachrichtigung.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`recipient`

Der Nachrichtenempfänger. Nur ein Empfänger wird unterstützt.

Erforderlich: `true`

Typ: `object` der die folgenden Eigenschaften beinhaltet:

`name`

Der Name des Empfängers.

Erforderlich: true

Typ: string

Gültiges Muster: .*

phone_number

Die Telefonnummer des Empfängers.

Erforderlich: true

Typ: string

Gültiges Muster: \+[1-9]+

message

Der Textinhalt der Textnachricht. Zu diesem Thema werden nur Textnachrichten unterstützt. Für Sprachnachrichten verwenden Sie `twilio/call`.

Erforderlich: true

Typ: string

Gültiges Muster: .+

from_number

Die Telefonnummer des Absenders. Twilio verwendet diese Telefonnummer, um die Nachricht zu senden. Diese Eigenschaft ist erforderlich, wenn der Parameter `DefaultFromPhoneNumber` nicht konfiguriert ist. Wenn `DefaultFromPhoneNumber` konfiguriert ist, können Sie diese Eigenschaft verwenden, um den Standard zu überschreiben.

Erforderlich: false

Typ: string

Gültiges Muster: \+[1-9]+

retries

Die Anzahl der Wiederholungen Der Standardwert ist 0.

Erforderlich: false

Typ: integer

id

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen.

Erforderlich: true

Typ: string

Gültiges Muster: .+

Beispieleingabe

```
{
  "request": {
    "recipient": {
      "name": "Darla",
      "phone_number": "+12345000000",
      "message": "Hello from the edge"
    },
    "from_number": "+19999999999",
    "retries": 3
  },
  "id": "request123"
}
```

Themenfilter: **twilio/call**

Nachrichten-Eigenschaften

request

Informationen über die Twilio-Benachrichtigung.

Erforderlich: true

Typ: object der die folgenden Eigenschaften beinhaltet:

recipient

Der Nachrichtenempfänger. Nur ein Empfänger wird unterstützt.

Erforderlich: `true`

Typ: object der die folgenden Eigenschaften beinhaltet:

`name`

Der Name des Empfängers.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

`phone_number`

Die Telefonnummer des Empfängers.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `\+[1-9]+`

`voice_message_location`

Die URL des Audioinhalts für die Sprachnachricht. Diese muss im TwiML-Format vorliegen. Zu diesem Thema werden nur Sprachnachrichten unterstützt. Für Textnachrichten verwenden Sie `twilio/txt`.

Erforderlich: `true`

Typ: `string`

Gültiges Muster: `.+`

`from_number`

Die Telefonnummer des Absenders. Twilio verwendet diese Telefonnummer, um die Nachricht zu senden. Diese Eigenschaft ist erforderlich, wenn der Parameter `DefaultFromPhoneNumber` nicht konfiguriert ist. Wenn `DefaultFromPhoneNumber` konfiguriert ist, können Sie diese Eigenschaft verwenden, um den Standard zu überschreiben.

Erforderlich: false

Typ: string

Gültiges Muster: \+[1-9]+

retries

Die Anzahl der Wiederholungen Der Standardwert ist 0.

Erforderlich: false

Typ: integer

id

Eine willkürliche ID für die Anforderung. Diese Eigenschaft wird verwendet, um eine Eingangsanforderung einer Ausgabeantwort zuzuordnen.

Erforderlich: true

Typ: string

Gültiges Muster: .+

Beispieleingabe

```
{
  "request": {
    "recipient": {
      "name": "Darla",
      "phone_number": "+12345000000",
      "voice_message_location": "https://some-public-TwiML"
    },
    "from_number": "+19999999999",
    "retries": 3
  },
  "id": "request123"
}
```

Ausgabedaten

Dieser Connector veröffentlicht Statusinformationen als Ausgabedaten im MQTT-Thema.

Themenfilter im Abonnement

twilio/message/status

Beispielausgabe: Erfolg

```
{
  "response": {
    "status": "success",
    "payload": {
      "from_number": "+19999999999",
      "messages": {
        "message_status": "queued",
        "to_number": "+12345000000",
        "name": "Darla"
      }
    }
  },
  "id": "request123"
}
```

Beispielausgabe: Fehler

```
{
  "response": {
    "status": "fail",
    "error_message": "Recipient name cannot be None",
    "error": "InvalidParameter",
    "payload": None
  },
  "id": "request123"
}
```

Die Eigenschaft `payload` in der Ausgabe ist die Antwort der Twilio-API beim Senden der Nachricht. Wenn der Konnektor erkennt, dass die Eingabedaten ungültig sind (z. B. kein erforderliches Eingabefeld angegeben ist), gibt der Konnektor einen Fehler zurück und setzt den Wert auf `None`. Im Folgenden finden Sie Beispiele für Nutzlasten:

```
{
  'from_number': '+19999999999',
  'messages': {
    'name': 'Darla',
```

```
    'to_number': '+12345000000',  
    'message_status': 'undelivered'  
  }  
}
```

```
{  
  'from_number': '+19999999999',  
  'messages': {  
    'name': 'Darla',  
    'to_number': '+12345000000',  
    'message_status': 'queued'  
  }  
}
```

Beispiel für die Verwendung

Verwenden Sie die folgenden allgemeinen Schritte, um eine Python-3.7-Lambda-Beispielfunktion einzurichten, mit der Sie den Konnektor ausprobieren können.

Note

Die [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#) Themen [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#) und enthalten end-to-end Schritte, die zeigen, wie Sie den Twilio Notifications Connector einrichten, bereitstellen und testen.

1. Stellen Sie sicher, dass Sie die [Anforderungen](#) für den Konnektor erfüllen.
2. Erstellen und veröffentlichen Sie eine Lambda-Funktion, die Eingabedaten an den Connector sendet.

Speichern Sie den [Beispielcode](#) als PY-Datei. Laden Sie das [AWS IoT Greengrass Core SDK für Python](#) herunter und entpacken Sie es. Erstellen Sie dann ein ZIP-Paket, das die PY-Datei und den Ordner greengrasssdk auf Stammebene enthält. Dieses Zip-Paket ist das Bereitstellungspaket, in das Sie AWS Lambda hochladen.

Nachdem Sie die Lambda-Funktion Python 3.7 erstellt haben, veröffentlichen Sie eine Funktionsversion und erstellen Sie einen Alias.

3. Konfigurieren Sie Ihre Greengrass-Gruppe.

- a. Fügen Sie die Lambda-Funktion mit ihrem Alias hinzu (empfohlen). Konfigurieren Sie den Lambda-Lebenszyklus als langlebig (oder "Pinned": true in der CLI).
 - b. Fügen Sie die erforderliche geheime Ressource hinzu und gewähren Sie der Lambda-Funktion Lesezugriff.
 - c. Fügen Sie den Konnektor hinzu und konfigurieren Sie seine [Parameter](#).
 - d. Fügen Sie Abonnements hinzu, die es dem Konnektor ermöglichen, [Eingabedaten](#) zu empfangen und [Ausgabedaten](#) zu unterstützten Themenfiltern zu senden.
 - Legen Sie die Lambda-Funktion als Quelle und den Konnektor als Ziel fest und verwenden Sie einen unterstützten Eingabethemenfilter.
 - Legen Sie den Konnektor als Quelle und AWS IoT Core als Ziel fest und verwenden Sie einen unterstützten Ausgabethemenfilter. Sie verwenden dieses Abonnement, um Statusmeldungen in der AWS IoT Konsole anzuzeigen.
4. Stellen Sie die Gruppe bereit.
 5. Abonnieren Sie in der AWS IoT Konsole auf der Testseite das Thema Ausgabedaten, um Statusmeldungen vom Connector anzuzeigen. Die Lambda-Beispielfunktion ist langlebig und beginnt unmittelbar nach der Bereitstellung der Gruppe mit dem Senden von Nachrichten.

Wenn Sie mit dem Testen fertig sind, können Sie den Lambda-Lebenszyklus auf On-Demand (oder "Pinned": false in der CLI) setzen und die Gruppe bereitstellen. Dadurch wird verhindert, dass die Funktion Nachrichten sendet.

Beispiel

Die folgende Lambda-Beispielfunktion sendet eine Eingabenachricht an den Konnektor. Dieses Beispiel löst eine Textnachricht aus.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():

    txt = {
```

```
        "request": {
            "recipient" : {
                "name": "Darla",
                "phone_number": "+12345000000",
                "message": 'Hello from the edge'
            },
            "from_number" : "+19999999999"
        },
        "id" : "request123"
    }

    print("Message To Publish: ", txt)

    client.publish(topic=TXT_INPUT_TOPIC,
                  payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Lizenzen

Der Twilio Notifications Connector umfasst die folgende Software/Lizenzierung von Drittanbietern:

- [twilio-python](#)/MIT

Dieser Connector ist im Rahmen der [Greengrass Core Software-Lizenzvereinbarung](#) veröffentlicht.

Änderungsprotokoll

In der folgenden Tabelle werden die Änderungen in den einzelnen Versionen des Connectors beschrieben.

Version	Änderungen
5	Der Parameter <code>IsolationMode</code> wurde hinzugefügt, um den Containerisierungsmodus für den Konnektor zu konfigurieren.

Version	Änderungen
4	Die Lambda-Laufzeit wurde auf Python 3.7 aktualisiert, was die Laufzeitanforderungen ändert.
3	Beheben, um übermäßige Protokollierung zu reduzieren.
2	Kleinere Fehlerbehebungen und Verbesserungen.
1	Erstversion.

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called “Aktualisieren von Konnektorversionen”](#).

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [the section called “Erste Schritte mit Konnektoren \(CLI\)”](#)
- [Twilio-API-Referenz](#)

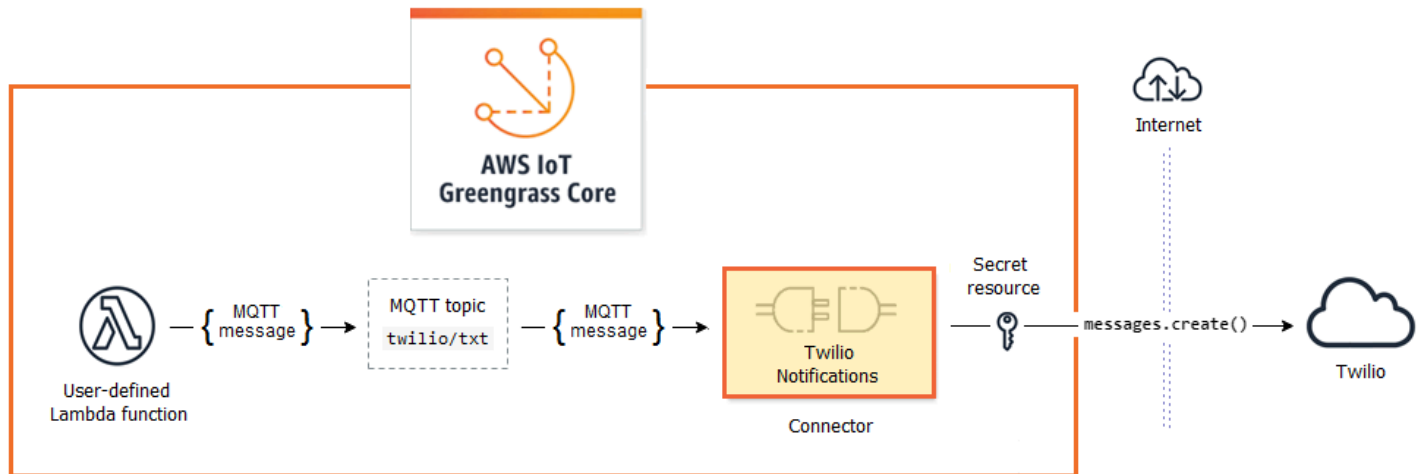
Erste Schritte mit Greengrass-Konnektoren (Konsole)

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Dieses Tutorial zeigt, wie Sie den verwenden, AWS-Managementkonsole um mit Anschlüssen zu arbeiten.

Verwenden Sie Konnektoren, um Ihren Entwicklungslebenszyklus zu beschleunigen. Konnektoren sind vorgefertigte, wiederverwendbare Module, die die Interaktion mit Services, Protokollen und Ressourcen erleichtern können. Sie können Ihnen helfen, die Geschäftslogik schneller auf Greengrass-Geräten zu implementieren. Weitere Informationen finden Sie unter [Integrieren von Services und Protokollen mit Konnektoren](#).

In diesem Tutorial konfigurieren und implementieren Sie den [Twilio Notifications](#) Connector. Der Konnektor empfängt Twilio-Nachrichteninformationen als Eingangsdaten und löst dann eine Twilio-SMS aus. Der Datenfluss ist in der folgenden Abbildung dargestellt.



Nachdem Sie den Connector konfiguriert haben, erstellen Sie eine Lambda-Funktion und ein Abonnement.

- Die Funktion wertet simulierte Daten eines Temperatursensors aus. Es veröffentlicht bedingt die Twilio-Nachrichteninformationen zu einem MQTT-Thema. Dies ist das Thema, das der Konnektor abonniert hat.
- Das Abonnement ermöglicht es der Funktion, zum Thema zu veröffentlichen, und dem Konnektor, die Daten vom Thema zu empfangen.

Der Twilio Notifications Connector benötigt ein Twilio-Authentifizierungstoken, um mit der Twilio-API zu interagieren. Das Token ist ein Geheimnis vom Typ Text, das in einer Gruppenressource erstellt wird, in der AWS Secrets Manager und von dort aus referenziert wird. Dadurch kann AWS IoT Greengrass eine lokale Kopie des Secrets auf dem Greengrass-Core erstellen, wo es verschlüsselt und dem Connector zur Verfügung gestellt wird. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Erstellen Sie ein Secrets Manager Manager-Geheimnis](#)
2. [Hinzufügen einer geheimen Ressource zu einer Gruppe](#)
3. [Hinzufügen eines Konnektors zur Gruppe](#)

4. [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#)
5. [Erstellen einer Lambda-Funktion](#)
6. [Hinzufügen einer Funktion zur Gruppe](#)
7. [Hinzufügen von Abonnements zur Gruppe](#)
8. [Bereitstellen der Gruppe](#)
9. [the section called “Testen der Lösung”](#)

Für dieses Tutorial benötigen Sie ungefähr 20 Minuten.

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Core (v1.9.3 oder höher). Weitere Informationen zum Erstellen einer Greengrass-Gruppe und Greengrass Core finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#). Das Tutorial „Erste Schritte“ enthält auch die Schritte zur Installation der AWS IoT Greengrass -Core Software.
- Python 3.7 ist auf dem AWS IoT Greengrass Core-Gerät installiert.
- AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit greengrass - beginnen.

- Eine Twilio-Konto-SID, ein Authent-Token und eine Twilio-aktivierte Telefonnummer. Nachdem Sie ein Twilio-Projekt erstellt haben, stehen diese Werte im Projekt-Dashboard zur Verfügung.

Note

Sie können ein Twilio-Testkonto verwenden. Wenn Sie ein Testkonto verwenden, müssen Sie Telefonnummern von Empfängern, die keine Twilio-Empfänger sind, zu einer Liste

verifizierter Telefonnummern hinzufügen. Weitere Informationen finden Sie unter [So arbeiten Sie mit Ihrem kostenlosen Twilio-Testkonto](#).

Schritt 1: Erstellen Sie ein Secrets Manager Manager-Geheimnis

In diesem Schritt verwenden Sie die AWS Secrets Manager Konsole, um ein Textgeheimnis für Ihr Twilio-Authentifizierungstoken zu erstellen.

1. Melden Sie sich bei der [AWS Secrets Manager -Konsole](#) an.

Note

Weitere Informationen zu diesem Vorgang finden Sie unter [Schritt 1: Ihr Geheimnis erstellen und speichern AWS Secrets Manager im AWS Secrets Manager Benutzerhandbuch](#).

2. Wählen Sie Store a new secret (Ein neues Secret speichern).
3. Wählen Sie unter Geheimtyp auswählen die Option Anderer Geheimtyp aus.
4. Geben Sie unter Geben Sie die key/value Paare an, die für dieses Geheimnis gespeichert werden sollen, auf der Registerkarte Klartext Ihr Twilio-Authentifizierungstoken ein. Entfernen Sie alle JSON-Formatierungen und geben Sie nur den Token-Wert ein.
5. Lassen Sie aws/secretsmanager für den Verschlüsselungsschlüssel ausgewählt und wählen Sie dann Weiter.

Note

Es fallen keine Gebühren an, AWS KMS wenn Sie den AWS verwalteten Standardschlüssel verwenden, den Secrets Manager in Ihrem Konto erstellt.

6. Geben Sie für Secret name (Secret-Name) **greengrass-TwilioAuthToken** ein und klicken Sie auf Next (Weiter).

 Note

Standardmäßig ermöglicht AWS IoT Greengrass die Greengrass-Dienstrolle das Abrufen des Werts von Geheimnissen mit Namen, die mit greengrass - beginnen. Weitere Informationen finden Sie unter [Anforderungen für Secrets](#).

7. Für dieses Tutorial ist keine Rotation erforderlich. Wählen Sie „Automatische Rotation deaktivieren“ und anschließend „Weiter“.
8. Prüfen Sie auf der Seite Review (Prüfen) Ihre Einstellungen und wählen Sie Store (Speichern).

Als nächstes erstellen Sie in Ihrer Greengrass-Gruppe eine geheime Ressource, die auf das Secret verweist.

Schritt 2: Hinzufügen einer geheimen Ressource zu einer Greengrass-Gruppe

In diesem Schritt fügen Sie eine geheime Ressource zur Greengrass-Gruppe hinzu. Diese Ressource ist eine Referenz auf das Secret, das Sie im vorherigen Schritt angelegt haben.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe, zu der Sie die geheime Ressource hinzufügen möchten.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Ressourcen und scrollen Sie dann nach unten zum Abschnitt Secrets. Im Bereich Secrets werden die geheimen Ressourcen angezeigt, die zu der Gruppe gehören. In diesem Bereich können Sie geheime Ressourcen hinzufügen, bearbeiten und entfernen.

 Note

Alternativ können Sie mit der Konsole eine geheime und geheime Ressource erstellen, wenn Sie einen Connector oder eine Lambda-Funktion konfigurieren. Sie können dies auf der Seite „Parameter konfigurieren“ des Connectors oder auf der Seite „Ressourcen“ der Lambda-Funktion tun.

4. Wählen Sie im Bereich Secrets die Option Hinzufügen aus.

5. Geben Sie auf der Seite Geheime Ressource hinzufügen **MyTwilioAuthToken** den Namen der Ressource ein.
6. Wählen Sie für das Secret greengrass- TwilioAuthToken.
7. Im Abschnitt Select labels (optional) steht das AWSCURRENT Staging-Label für die neueste Version des Secrets. Dieses Label ist immer in einer geheimen Ressource enthalten.

 Note

Für dieses Tutorial ist nur das AWSCURRENT Label erforderlich. Sie können optional Beschriftungen hinzufügen, die für Ihre Lambda-Funktion oder Ihren Lambda-Konnektor erforderlich sind.

8. Wählen Sie Add resource (Ressource hinzufügen) aus.

Schritt 3: Hinzufügen eines Konnektors zur Greengrass-Gruppe

In diesem Schritt konfigurieren Sie Parameter für den [Twilio Notifications Connector](#) und fügen ihn der Gruppe hinzu.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Option Connectors (Konnektoren), und wählen Sie dann Add a connector (Konnektor hinzufügen).
2. Wählen Sie auf der Seite „Connector hinzufügen“ die Option Twilio Notifications aus.
3. Wählen Sie die -Version aus.
4. Gehen Sie im Abschnitt Konfiguration wie folgt vor:
 - Geben Sie für die Twilio-Authentifizierungstoken-Ressource die Ressource ein, die Sie im vorherigen Schritt erstellt haben.

 Note

Wenn Sie die Ressource eingeben, wird der ARN der geheimen Eigenschaft des Twilio-Authentifizierungstokens für Sie aufgefüllt.

- Geben Sie unter Default from phone number (Standard Telefonnummer) Ihre Twilio-aktivierte Telefonnummer ein.
- Geben Sie für Twilio account SID (Twilio-Konto-SID) Ihre Twilio-Konto-SID ein.

5. Wählen Sie Add resource (Ressource hinzufügen) aus.

Schritt 4: Erstellen Sie ein Lambda-Funktionsbereitstellungspaket

Um eine Lambda-Funktion zu erstellen, müssen Sie zunächst ein Lambda-Funktionsbereitstellungspaket erstellen, das den Funktionscode und die Abhängigkeiten enthält. Greengrass Lambda-Funktionen benötigen das [AWS IoT Greengrass Core SDK](#) für Aufgaben wie die Kommunikation mit MQTT-Nachrichten in der Kernumgebung und den Zugriff auf lokale Geheimnisse. In diesem Tutorial wird eine Python-Funktion erstellt, sodass Sie die Python-Version des SDK im Bereitstellungspaket verwenden.

1. Laden Sie von der [AWS IoT Greengrass Core SDK-Downloadseite](#) das AWS IoT Greengrass Core SDK für Python auf Ihren Computer herunter.
2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der greengrasssdk-Ordner.
3. Speichern Sie die folgende Pythoncode-Funktion in einer lokalen Datei namens `temp_monitor.py`.

```
import greengrasssdk
import json
import random

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

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
```

```
to_name = event['to_name']
to_number = event['to_number']
temp_report = 'temperature:' + str(event['temperature'])

return {
    "request": {
        "recipient": {
            "name": to_name,
            "phone_number": to_number,
            "message": temp_report
        }
    },
    "id": "request_" + str(random.randint(1,101))
}
```

4. Packen Sie die folgenden Elemente in einer ZIP-Datei mit dem Namen `temp_monitor_python.zip`. Verwenden Sie zum Erstellen der ZIP-Datei nur den Code und die entsprechenden Abhängigkeiten und nicht den dazugehörigen Ordner.
 - `temp_monitor.py`. App-Logik.
 - `greengrasssdk`. Erforderliche Bibliothek für Python Greengrass Lambda-Funktionen, die MQTT-Nachrichten veröffentlichen.

Dies ist Ihr Lambda-Funktionsbereitstellungspaket.

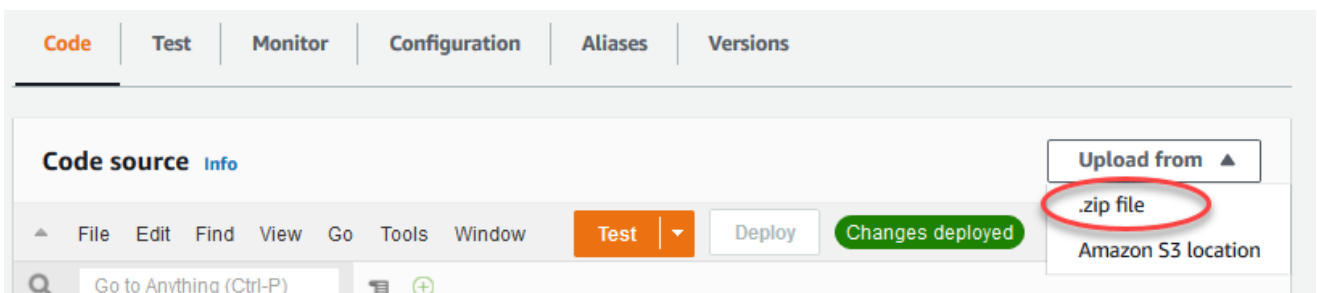
Erstellen Sie nun eine Lambda-Funktion, die das Bereitstellungspaket verwendet.

Schritt 5: Erstellen Sie eine Lambda-Funktion in der Konsole AWS Lambda

In diesem Schritt verwenden Sie die AWS Lambda Konsole, um eine Lambda-Funktion zu erstellen und sie für die Verwendung Ihres Bereitstellungspakets zu konfigurieren. Anschließend veröffentlichen Sie eine Funktionsversion und erstellen einen Alias.

1. Erstellen Sie zunächst die Lambda-Funktion.
 - a. Wählen Sie in der AWS-Managementkonsole Dienste aus und öffnen Sie die AWS Lambda Konsole.
 - b. Wählen Sie „Funktion erstellen“ und anschließend „Von Grund auf neu erstellen“.
 - c. Verwenden Sie im Abschnitt Basic information (Basisinformationen) folgende Werte:

- Geben Sie für Function name (Funktionsname) **TempMonitor** ein.
 - Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Behalten Sie für Berechtigungen die Standardeinstellung bei. Dadurch wird eine Ausführungsrolle erstellt, die grundlegende Lambda-Berechtigungen gewährt. Diese Rolle wird nicht verwendet von AWS IoT Greengrass.
- d. Klicken Sie unten auf der Seite auf Create function.
2. Registrieren Sie als Nächstes den Handler und laden Sie Ihr Lambda-Funktionsbereitstellungspaket hoch.
- a. Wählen Sie auf der Registerkarte Code unter Codequelle die Option Upload von aus. Wählen Sie in der Dropdownliste die ZIP-Datei aus.



- b. Wählen Sie Hochladen und dann Ihr `temp_monitor_python.zip` Bereitstellungspaket aus. Wählen Sie dann Save (Speichern) aus.
- c. Wählen Sie auf der Registerkarte Code für die Funktion unter Laufzeiteinstellungen die Option Bearbeiten aus, und geben Sie dann die folgenden Werte ein.
- Wählen Sie für Runtime (Laufzeit) die Option Python 3.7 aus.
 - Geben Sie unter Handler **`temp_monitor.function_handler`** ein.
- d. Wählen Sie Speichern.

Note

Die Testtaste auf der AWS Lambda Konsole funktioniert mit dieser Funktion nicht. Das AWS IoT Greengrass Core SDK enthält keine Module, die erforderlich sind, um Ihre Greengrass Lambda-Funktionen unabhängig in der AWS Lambda Konsole auszuführen. Diese Module (zum Beispiel `greengrass_common`) werden den Funktionen zur Verfügung gestellt, nachdem sie auf Ihrem Greengrass-Kern bereitgestellt wurden.

3. Veröffentlichen Sie jetzt die erste Version Ihrer Lambda-Funktion und erstellen Sie einen [Alias für die Version](#).

 Note

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

- a. Wählen Sie im Menü Actions die Option Publish new version aus.
- b. Geben Sie unter Version description (Versionsbeschreibung) den Wert **First version** ein und wählen Sie dann Publish (Veröffentlichen) aus.
- c. Wählen Sie auf der Konfigurationsseite TempMonitor: 1 im Menü Aktionen die Option Alias erstellen aus.
- d. Geben Sie auf der Seite Create a new alias folgende Werte an:
 - Geben Sie unter Name **GG_TempMonitor** ein.
 - Wählen Sie für Version die Option 1.

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

- e. Wählen Sie Erstellen aus.

Jetzt sind Sie bereit, die Lambda-Funktion zu Ihrer Greengrass-Gruppe hinzuzufügen.

Schritt 6: Fügen Sie der Greengrass-Gruppe eine Lambda-Funktion hinzu

In diesem Schritt fügen Sie der Gruppe die Lambda-Funktion hinzu und konfigurieren dann ihre Lebenszyklus- und Umgebungsvariablen. Weitere Informationen finden Sie unter [the section called "Steuerung der Ausführung von Greengrass Lambda-Funktionen"](#).

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Lambda-Funktionen aus.

2. Wählen Sie unter Meine Lambda-Funktionen die Option Hinzufügen aus.
3. Wählen TempMonitorSie auf der Seite Lambda-Funktion hinzufügen Ihre Lambda-Funktion aus.
4. Wählen Sie für die Lambda-Funktionsversion Alias: TempMonitor GG_.
5. Wählen Sie Lambda-Funktion hinzufügen.

Schritt 7: Hinzufügen von Abonnements zur Greengrass-Gruppe

In diesem Schritt fügen Sie ein Abonnement hinzu, das es der Lambda-Funktion ermöglicht, Eingabedaten an den Connector zu senden. Der Konnektor definiert die MQTT-Themen, die er abonniert, und dieses Abonnement verwendet eines dieser Themen. Es ist das gleiche Thema, zu dem die Beispielfunktion veröffentlicht.

Für dieses Tutorial erstellen Sie auch Abonnements, die es der Funktion ermöglichen, simulierte Temperaturmesswerte vom Connector AWS IoT zu empfangen AWS IoT und Statusinformationen vom Connector zu empfangen.

1. Wählen Sie auf der Gruppenkonfigurationsseite die Registerkarte Abonnements und dann Abonnement hinzufügen aus.
2. Konfigurieren Sie auf der Seite Abonnement erstellen die Quelle und das Ziel wie folgt:
 - a. Wählen Sie als Quelltyp die Option Lambda-Funktion und wählen Sie TempMonitordann.
 - b. Wählen Sie als Zieltyp die Option Connector und dann Twilio Notifications aus.
3. Wählen **twilio/txt** Sie für den Themenfilter.
4. Wählen Sie Create subscription (Abonnement erstellen) aus.
5. Wiederholen Sie die Schritte 1 bis 4, um ein Abonnement AWS IoT zu erstellen, mit dem Sie Nachrichten für die Funktion veröffentlichen können.
 - a. Wählen Sie als Quelltyp Service und dann IoT Cloud aus.
 - b. Wählen Sie für Select a target die Option Lambda-Funktion aus, und wählen Sie TempMonitordann.
 - c. Geben Sie für Topic filter (Themenfilter) die Zeichenfolge **temperature/input** ein.
6. Wiederholen Sie die Schritte 1 - 4, um ein Abonnement zu erstellen, das es dem Konnektor ermöglicht, Nachrichten an AWS IoT zu veröffentlichen.
 - a. Wählen Sie als Quelltyp die Option Connector und dann Twilio Notifications aus.

- b. Wählen Sie als Zieltyp Service und dann IoT Cloud aus.
- c. Für Themenfilter wird automatisch **twilio/message/status** eingegeben. Dies ist das vordefinierte Thema, in das der Konnektor veröffentlicht.

Schritt 8: Bereitstellen der Greengrass-Gruppe

Stellen Sie die Gruppe auf dem Core-Gerät bereit.

1. Stellen Sie sicher, dass der AWS IoT Greengrass Core läuft. Führen Sie im Raspberry Pi-Terminal die folgenden Befehle aus, falls nötig.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen root-Eintrag für `/greengrass/ggc/packages/ggc-version/bin/daemon` enthält, dann wird der Daemon ausgeführt.

Note

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Deploy aus.
3.
 - a. Wählen Sie auf der Registerkarte Lambda-Funktionen im Abschnitt System-Lambda-Funktionen die Option IP-Detektor und dann Bearbeiten aus.
 - b. Wählen Sie im Dialogfeld IP-Detektoreinstellungen bearbeiten die Option MQTT-Broker-Endpunkte automatisch erkennen und überschreiben aus.
 - c. Wählen Sie Speichern.

Damit können Geräte automatisch Core-Verbindungsinformationen abrufen, z. B. die IP-Adresse, DNS und die Portnummer. Eine automatische Erkennung wird empfohlen,

unterstützt aber AWS IoT Greengrass auch manuell angegebene Endpunkte. Sie werden nur bei der ersten Bereitstellung der Gruppe zur Angabe der Ermittlungsmethode aufgefordert.

 Note

Wenn Sie dazu aufgefordert werden, erteilen Sie die Erlaubnis, die [Greengrass-Servicerolle zu erstellen und sie Ihrer AWS-Konto aktuellen AWS-Region Rolle zuzuordnen](#). Diese Rolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Diensten.

Auf der Seite Deployments werden der Zeitstempel, die Versions-ID und der Status der Bereitstellung angegeben. Nach Abschluss sollte der für die Bereitstellung angezeigte Status Abgeschlossen lauten.

Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

 Note

Eine Greengrass-Gruppe kann jeweils nur eine Version des Connectors enthalten. Weitere Informationen zum Aktualisieren einer Konnektorversion finden Sie unter [the section called "Aktualisieren von Konnektorversionen"](#).

Testen der Lösung

1. Wählen Sie auf der Startseite der AWS IoT Konsole die Option Test aus.
2. Verwenden Sie für Thema abonnieren die folgenden Werte und wählen Sie dann Abonnieren aus. Der Twilio Notifications Connector veröffentlicht Statusinformationen zu diesem Thema.

Eigenschaft	Wert
Abonnementthema	twilio/message/status
MQTT-Nutzlast-Anzeige	Zeigt Nutzlasten als Zeichenfolgen an

3. Verwenden Sie für In Thema veröffentlichen die folgenden Werte und wählen Sie dann Veröffentlichen, um die Funktion aufzurufen.

Eigenschaft	Wert
Thema	Temperatur/Input
Fehlermeldung	<i>recipient-name</i> Ersetzen Sie es <i>recipient-phone-number</i> durch einen Namen und die Telefonnummer des SMS- Empfängers. Beispiel: +12345000000 <pre>{ "to_name": " <i>recipient-name</i> ", "to_number": " <i>recipient-phone-number</i> ", "temperature": 31 }</pre> Wenn Sie ein Testkonto verwenden, müssen Sie einer Liste verifizierter Telefonnummern hinzufügen, die keine Twilio-Empfänger sind. Weitere Informationen finden Sie unter Verifizieren Sie Ihre persönliche Telefonnummer.

Wenn erfolgreich, erhält der Empfänger die Textnachricht und die Konsole zeigt den Status success für die [Ausgangsdaten](#).

Ändern Sie nun den temperature in der Eingabemeldung auf **29** und veröffentlichen Sie. Da dieser Wert unter 30 liegt, löst die TempMonitor Funktion keine Twilio-Nachricht aus.

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#)

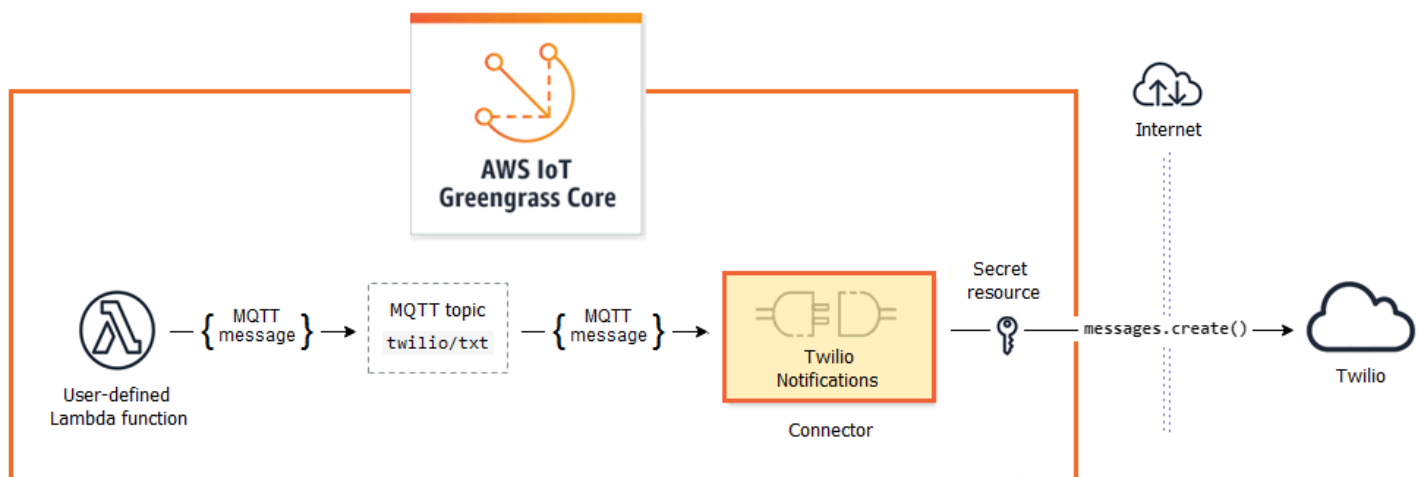
Erste Schritte mit Greengrass-Konnektoren (CLI)

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

Dieses Tutorial zeigt, wie Sie mit Konnektoren arbeiten. AWS CLI

Verwenden Sie Konnektoren, um Ihren Entwicklungslebenszyklus zu beschleunigen. Konnektoren sind vorgefertigte, wiederverwendbare Module, die die Interaktion mit Services, Protokollen und Ressourcen erleichtern können. Sie können Ihnen helfen, die Geschäftslogik schneller auf Greengrass-Geräten zu implementieren. Weitere Informationen finden Sie unter [Integrieren von Services und Protokollen mit Konnektoren](#).

In diesem Tutorial konfigurieren und implementieren Sie den [Twilio Notifications](#) Connector. Der Konnektor empfängt Twilio-Nachrichteninformationen als Eingangsdaten und löst dann eine Twilio-SMS aus. Der Datenfluss ist in der folgenden Abbildung dargestellt.



Nachdem Sie den Connector konfiguriert haben, erstellen Sie eine Lambda-Funktion und ein Abonnement.

- Die Funktion wertet simulierte Daten eines Temperatursensors aus. Es veröffentlicht bedingt die Twilio-Nachrichteninformationen zu einem MQTT-Thema. Dies ist das Thema, dass der Konnektor abonniert hat.
- Das Abonnement ermöglicht es der Funktion, zum Thema zu veröffentlichen, und dem Konnektor, die Daten vom Thema zu empfangen.

Der Twilio Notifications Connector benötigt ein Twilio-Authentifizierungstoken, um mit der Twilio-API zu interagieren. Das Token ist ein Geheimnis vom Typ Text, das in einer Gruppenressource erstellt wird. AWS Secrets Manager und von dort aus referenziert wird. Dadurch kann AWS IoT Greengrass eine lokale Kopie des Secrets auf dem Greengrass-Core erstellt werden, wo es verschlüsselt und dem Connector zur Verfügung gestellt wird. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Das Tutorial enthält die folgenden allgemeinen Schritte:

1. [Ein Secrets Manager Manager-Geheimnis erstellen](#)
2. [Erstellen einer Ressourcendefinition und -version](#)
3. [Erstellen einer Konnektordefinition und -version](#)
4. [Erstellen Sie ein Bereitstellungspaket für Lambda-Funktionen](#)
5. [Erstellen einer Lambda-Funktion](#)
6. [Erstellen einer Funktionsdefinition und -version](#)
7. [Erstellen einer Abonnementdefinition und -version](#)
8. [Erstellen einer Gruppenversion](#)
9. [Eine Bereitstellung auswählen](#)
10. [the section called "Testen der Lösung"](#)

Für dieses Tutorial benötigen Sie ungefähr 30 Minuten.

Mithilfe der API AWS IoT Greengrass

Es ist hilfreich, die folgenden Muster zu verstehen, wenn Sie mit Greengrass-Gruppen und -Gruppenkomponenten arbeiten (z. B. die Konnektoren, Funktionen und Ressourcen in der Gruppe).

- An der Spitze der Hierarchie befindet sich ein Definition-Objekt, das ein Container für Version-Objekte ist. Eine Version wiederum ist ein Container für die Konnektoren, Funktionen oder andere Komponententypen.
- Wenn Sie im Greengrass Core bereitstellen, stellen Sie eine bestimmte Gruppenversion bereit. Eine Gruppenversion kann eine Version jeder Art von Komponente enthalten. Ein Core ist erforderlich, aber die anderen werden bei Bedarf einbezogen.
- Versionen sind unveränderlich, daher müssen Sie neue Versionen anlegen, wenn Sie Änderungen vornehmen möchten.

 Tip

Wenn Sie bei der Ausführung eines AWS CLI Befehls eine Fehlermeldung erhalten, fügen Sie den `--debug` Parameter hinzu und führen Sie den Befehl erneut aus, um weitere Informationen zu dem Fehler zu erhalten.

Mit der AWS IoT Greengrass API können Sie mehrere Definitionen für einen Komponententyp erstellen. Beispielsweise können Sie jedes Mal, wenn Sie ein `FunctionDefinition`-Objekt erstellen, ein `FunctionDefinitionVersion` erstellen, oder Sie können neue Versionen zu einer bestehenden Definition hinzufügen. Diese Flexibilität ermöglicht es Ihnen, Ihr Versionsmanagementsystem individuell anzupassen.

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Eine Greengrass-Gruppe und ein Greengrass-Core (v1.9.3 oder höher). Weitere Informationen zum Erstellen einer Greengrass-Gruppe und Greengrass Core finden Sie unter [Erste Schritte mit AWS IoT Greengrass](#). Das Tutorial „Erste Schritte“ enthält auch die Schritte zur Installation der AWS IoT Greengrass -Core Software.
- Python 3.7 ist auf dem AWS IoT Greengrass Core-Gerät installiert.
- AWS IoT Greengrass muss so konfiguriert sein, dass es lokale Geheimnisse unterstützt, wie in [Secrets Requirements](#) beschrieben.

 Note

Diese Anforderung beinhaltet die Gewährung des Zugriffs auf Ihre Secrets Manager Manager-Geheimnisse. Wenn Sie die standardmäßige Greengrass-Servicerolle verwenden, ist Greengrass berechtigt, die Werte von Geheimnissen abzurufen, deren Namen mit `greengrass -` beginnen.

- Eine Twilio-Konto-SID, ein Authent-Token und eine Twilio-aktivierte Telefonnummer. Nachdem Sie ein Twilio-Projekt erstellt haben, stehen diese Werte im Projekt-Dashboard zur Verfügung.

 Note

Sie können ein Twilio-Testkonto verwenden. Wenn Sie ein Testkonto verwenden, müssen Sie einer Liste verifizierter Telefonnummern hinzufügen, die keine Twilio-Empfänger sind. Weitere Informationen finden Sie unter [So arbeiten Sie mit Ihrem kostenlosen Twilio-Testkonto](#).

- AWS CLI auf Ihrem Computer installiert und konfiguriert. Weitere Informationen finden Sie [unter Installation AWS Command Line Interface](#) und [Konfiguration von AWS CLI im AWS Command Line Interface](#) Benutzerhandbuch.

Die Beispiele in diesem Tutorial sind für Linux und andere Unix-basierte Systeme geschrieben. Wenn Sie Windows verwenden, finden Sie unter [Angaben von Parameterwerten für die Informationen AWS Command Line Interface](#) zu Unterschieden in der Syntax.

Wenn der Befehl eine JSON-Zeichenkette enthält, zeigt das Tutorial ein Beispiel, das das JSON auf einer einzigen Zeile hat. Auf einigen Systemen ist es möglicherweise einfacher, Befehle in diesem Format zu bearbeiten und auszuführen.

Schritt 1: Erstellen Sie ein Secrets Manager Manager-Geheimnis

In diesem Schritt verwenden Sie die AWS Secrets Manager API, um ein Geheimnis für Ihr Twilio-Authentifizierungstoken zu erstellen.

1. Erstellen Sie zunächst das Secret.

- Ersetzen Sie es *twilio-auth-token* durch Ihr Twilio-Authentifizierungstoken.

```
aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-string twilio-auth-token
```

 Note

Standardmäßig ermöglicht AWS IoT Greengrass die Greengrass-Dienstrolle das Abrufen des Werts von Geheimnissen mit Namen, die mit greengrass - beginnen. Weitere Informationen finden Sie unter [Anforderungen für Secrets](#).

2. Kopieren Sie den ARN des Secrets aus der Ausgabe. Sie verwenden dies, um die geheime Ressource zu erstellen und den Twilio Notifications Connector zu konfigurieren.

Schritt 2: Erstellen einer Ressourcendefinition und -version

In diesem Schritt verwenden Sie die AWS IoT Greengrass API, um eine geheime Ressource für Ihr Secrets Manager-Geheimnis zu erstellen.

1. Erstellen Sie eine Ressourcendefinition, die eine Initialversion enthält.
 - ***secret-arn*** Ersetzen Sie es durch das Geheimnis, das Sie im vorherigen Schritt kopiert haben. ARN

JSON Expanded

```
aws greengrass create-resource-definition --name MyGreengrassResources --
initial-version '{
  "Resources": [
    {
      "Id": "TwilioAuthToken",
      "Name": "MyTwilioAuthToken",
      "ResourceDataContainer": {
        "SecretsManagerSecretResourceData": {
          "ARN": "secret-arn"
        }
      }
    }
  ]
}'
```

JSON Single-line

```
aws greengrass create-resource-definition \  
--name MyGreengrassResources \  
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",  
  "Name": "MyTwilioAuthToken", "ResourceDataContainer":  
    {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. Kopieren Sie den LatestVersionArn der Ressourcendefinition aus der Ausgabe. Mit diesem Wert fügen Sie die Version der Ressourcendefinition der Gruppenversion hinzu, die Sie im Core bereitstellen.

Schritt 3: Erstellen einer Konnektordefinition und -version

In diesem Schritt konfigurieren Sie Parameter für den Twilio Notifications Connector.

1. Erstellen Sie eine Konnektordefinition mit einer Initialversion.
 - Ersetzen Sie es *account-sid* durch die SID Ihres Twilio-Kontos.
 - *secret-arn* Ersetzen Sie es durch das ARN Ihres Secrets Manager Manager-Geheimnisses. Der Konnektor verwendet dies, um den Wert des lokalen Secrets zu erhalten.
 - Ersetze es *phone-number* durch deine Twilio-fähige Telefonnummer. Twilio verwendet sie, um die Textnachricht zu initiieren. Dies kann in der Nutzlast der Eingabenachricht überschrieben werden. Verwenden Sie das folgende Format: +19999999999.

JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --  
initial-version '{  
  "Connectors": [  
    {  
      "Id": "MyTwilioNotificationsConnector",  
      "ConnectorArn": "arn:aws:greengrass:region::/connectors/  
TwilioNotifications/versions/4",  
      "Parameters": {  
        "TWILIO_ACCOUNT_SID": "account-sid",  
        "TwilioAuthTokenSecretArn": "secret-arn",
```

```

        "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
        "DefaultFromPhoneNumber": "phone-number"
    }
}
]
}'

```

JSON Single-line

```

aws greengrass create-connector-definition \
--name MyGreengrassConnectors \
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",
  "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",
  "TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}}]}'

```

Note

TwilioAuthToken ist die ID, die Sie im vorherigen Schritt zum Erstellen der geheimen Ressource verwendet haben.

2. Kopieren Sie den LatestVersionArn der Konnektordefinition aus der Ausgabe. Mit diesem Wert fügen Sie die Version der Konnektordefinition der Gruppenversion hinzu, die Sie im Core bereitstellen.

Schritt 4: Erstellen Sie ein Lambda-Funktionsbereitstellungspaket

Um eine Lambda-Funktion zu erstellen, müssen Sie zunächst ein Lambda-Funktionsbereitstellungspaket erstellen, das den Funktionscode und die Abhängigkeiten enthält. Greengrass Lambda-Funktionen benötigen das [AWS IoT Greengrass Core SDK](#) für Aufgaben wie die Kommunikation mit MQTT-Nachrichten in der Kernumgebung und den Zugriff auf lokale Geheimnisse. In diesem Tutorial wird eine Python-Funktion erstellt, sodass Sie die Python-Version des SDK im Bereitstellungspaket verwenden.

1. Laden Sie von der [AWS IoT Greengrass Core SDK-Downloadseite](#) das AWS IoT Greengrass Core SDK für Python auf Ihren Computer herunter.

2. Entpacken Sie das heruntergeladene Paket, um das SDK zu erhalten. Das SDK ist der greengrasssdk-Ordner.
3. Speichern Sie die folgende Pythoncode-Funktion in einer lokalen Datei namens temp_monitor.py.

```
import greengrasssdk
import json
import random

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

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```


4. Packen Sie die folgenden Elemente in einer ZIP-Datei mit dem Namen `temp_monitor_python.zip`. Verwenden Sie zum Erstellen der ZIP-Datei nur den Code und die entsprechenden Abhängigkeiten und nicht den dazugehörigen Ordner.
 - `temp_monitor.py`. App-Logik.
 - `greengrasssdk`. Erforderliche Bibliothek für Python Greengrass Lambda-Funktionen, die MQTT-Nachrichten veröffentlichen.

Dies ist Ihr Lambda-Funktionsbereitstellungspaket.

Schritt 5: Erstellen Sie eine Lambda-Funktion

Erstellen Sie nun eine Lambda-Funktion, die das Bereitstellungspaket verwendet.

1. Erstellen Sie eine IAM-Rolle, damit Sie den Rollen-ARN bei der Erstellung der Funktion übergeben können.

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
"2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
"lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

 Note

AWS IoT Greengrass verwendet diese Rolle nicht, da die Berechtigungen für Ihre Greengrass Lambda-Funktionen in der Greengrass-Gruppenrolle angegeben sind. Für dieses Tutorial erstellen Sie eine leere Rolle.

2. Kopieren Sie die Arn aus der Ausgabe.
3. Verwenden Sie die AWS Lambda API, um die Funktion zu erstellen. TempMonitor Der folgende Befehl geht davon aus, dass sich die Zip-Datei im aktuellen Verzeichnis befindet.
 - Ersetzen Sie *role-arn* durch den Arn, den Sie kopiert haben.

```
aws lambda create-function \  
--function-name TempMonitor \  
--zip-file fileb://temp_monitor_python.zip \  
--role role-arn \  
--handler temp_monitor.function_handler \  
--runtime python3.7
```

4. Veröffentlichen einer Version der Funktion.

```
aws lambda publish-version --function-name TempMonitor --description 'First  
version'
```

5. Erstellen Sie einen Alias für die veröffentlichte Version.

Greengrass-Gruppen können eine Lambda-Funktion nach Alias (empfohlen) oder nach Version referenzieren. Die Verwendung eines Alias erleichtert die Verwaltung von Codeaktualisierungen, da Sie Ihre Abonnementtabelle oder Gruppendefinition nicht ändern müssen, wenn der Funktionscode aktualisiert wird. Stattdessen verweisen Sie einfach mit dem Alias auf die neue Funktionsversion.

 Note

AWS IoT Greengrass unterstützt keine Lambda-Aliase für \$LATEST-Versionen.

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --  
function-version 1
```

6. Kopieren Sie die `AliasArn` aus der Ausgabe. Sie verwenden diesen Wert, wenn Sie die Funktion für ein Abonnement konfigurieren AWS IoT Greengrass und wenn Sie ein Abonnement erstellen.

Jetzt sind Sie bereit, die Funktion für zu konfigurieren AWS IoT Greengrass.

Schritt 6: Erstellen einer Funktionsdefinition und -version

Um eine Lambda-Funktion auf einem AWS IoT Greengrass Core zu verwenden, erstellen Sie eine Funktionsdefinitionsversion, die per Alias auf die Lambda-Funktion verweist und die Konfiguration auf Gruppenebene definiert. Weitere Informationen finden Sie unter [the section called “Steuerung der Ausführung von Greengrass Lambda-Funktionen”](#).

1. Erstellen Sie eine Funktionsdefinition, die eine Initialversion enthält.
 - Ersetzen Sie sie `alias-arn` durch die DateiAliasArn, die Sie bei der Erstellung des Alias kopiert haben.

JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --  
initial-version '{  
  "Functions": [  
    {  
      "Id": "TempMonitorFunction",  
      "FunctionArn": "alias-arn",  
      "FunctionConfiguration": {  
        "Executable": "temp_monitor.function_handler",  
        "MemorySize": 16000,  
        "Timeout": 5  
      }  
    }  
  ]  
}
```

```
}'
```

JSON Single-line

```
aws greengrass create-function-definition \  
--name MyGreengrassFunctions \  
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",  
  "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":  
    "temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. Kopieren Sie die LatestVersionArn aus der Ausgabe. Mit diesem Wert fügen Sie die Version der Funktionsdefinition der Gruppenversion hinzu, die Sie im Core bereitstellen.
3. Kopieren Sie die Id aus der Ausgabe. Diesen Wert verwenden Sie später, wenn Sie die Funktion aktualisieren.

Schritt 7: Erstellen einer Abonnementdefinition und -version

In diesem Schritt fügen Sie ein Abonnement hinzu, das es der Lambda-Funktion ermöglicht, Eingabedaten an den Connector zu senden. Der Konnektor definiert die MQTT-Themen, die er abonniert, und dieses Abonnement verwendet eines dieser Themen. Es ist das gleiche Thema, zu dem die Beispielfunktion veröffentlicht.

Für dieses Tutorial erstellen Sie auch Abonnements, die es der Funktion ermöglichen, simulierte Temperaturmesswerte vom Connector AWS IoT zu empfangen AWS IoT und Statusinformationen vom Connector zu empfangen.

1. Erstellen Sie eine Abonnementdefinition, die eine Initialversion enthält, die die Abonnements umfasst.
 - *alias-arn* Ersetzen Sie es durch AliasArn das, was Sie kopiert haben, als Sie den Alias für die Funktion erstellt haben. Verwenden Sie diesen ARN für beide Abonnements, die ihn verwenden.

JSON Expanded

```
aws greengrass create-subscription-definition --initial-version '{  
  "Subscriptions": [  

```

```

    {
      "Id": "TriggerNotification",
      "Source": "alias-arn",
      "Subject": "twilio/txt",
      "Target": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4"
    },
    {
      "Id": "TemperatureInput",
      "Source": "cloud",
      "Subject": "temperature/input",
      "Target": "alias-arn"
    },
    {
      "Id": "OutputStatus",
      "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
      "Subject": "twilio/message/status",
      "Target": "cloud"
    }
  ]
}'

```

JSON Single-line

```

aws greengrass create-subscription-definition \
--initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
"alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
connectors/TwilioNotifications/versions/4"}, {"Id": "TemperatureInput",
"Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":
"cloud"}]}'

```

2. Kopieren Sie die LatestVersionArn aus der Ausgabe. Mit diesem Wert fügen Sie die Version der Abonnementdefinition der Gruppenversion hinzu, die Sie im Core bereitstellen.

Schritt 8: Erstellen einer Gruppenversion

Jetzt können Sie eine Gruppenversion erstellen, die alle Elemente enthält, die Sie bereitstellen möchten. Dazu legen Sie eine Gruppenversion an, die auf die Zielversion jedes Komponententyps verweist.

Zuerst erhalten Sie die Gruppen-ID und den ARN der Core-Definitionsversion. Diese Werte sind erforderlich, um die Gruppenversion zu erstellen.

1. Rufen Sie die ID der Gruppe und die neueste Gruppenversion ab:

- a. Rufen Sie IDs die Greengrass-Zielgruppe und die Gruppenversion ab. Bei diesem Verfahren wird davon ausgegangen, dass es sich um die neueste Gruppen- und Gruppenversion handelt. Die folgende Abfrage gibt die zuletzt erstellte Gruppe zurück.

```
aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))[0]"
```

Sie können auch nach Namen abfragen. Gruppennamen müssen nicht eindeutig sein, sodass mehrere Gruppen zurückgegeben werden können.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

 Note

Sie finden diese Werte auch in der AWS IoT Konsole. Die Gruppen-ID wird auf der Seite Einstellungen der Gruppe angezeigt. Die Gruppenversion IDs wird auf der Registerkarte Bereitstellungen der Gruppe angezeigt.

- b. Kopieren Sie die Id der Zielgruppe aus der Ausgabe. Sie verwenden sie, um die Core-Definitionsversion zu erhalten, und wenn Sie die Gruppe bereitstellen.
 - c. Kopieren Sie die LatestVersion aus der Ausgabe, d. h. die ID der letzten Version, die der Gruppe hinzugefügt wurde. Damit erhalten Sie die Core-Definitionsversion.
2. Abrufen des ARN der Core-Definitionsversion:
- a. Abrufen der Gruppenversion. Für diesen Schritt gehen wir davon aus, dass die neueste Gruppenversion eine Core-Definitionsversion enthält.
 - **group-id** Ersetzen Sie es durch Id das, was Sie für die Gruppe kopiert haben.
 - **group-version-id** Ersetzen Sie es durch LatestVersion das, was Sie für die Gruppe kopiert haben.

```
aws greengrass get-group-version \
```

```
--group-id group-id \  
--group-version-id group-version-id
```

b. Kopieren Sie die `CoreDefinitionVersionArn` aus der Ausgabe.

3. Erstellen einer Gruppenversion

- *group-id* Ersetzen Sie es durch Id das, was Sie für die Gruppe kopiert haben.
- *core-definition-version-arn* Ersetzen Sie es durch `CoreDefinitionVersionArn` das, was Sie für die Version der Kerndefinition kopiert haben.
- *resource-definition-version-arn* Ersetzen Sie es durch `LatestVersionArn` das, was Sie für die Ressourcendefinition kopiert haben.
- *connector-definition-version-arn* Ersetzen Sie es durch `LatestVersionArn` das, was Sie für die Konnektordefinition kopiert haben.
- *function-definition-version-arn* Ersetzen Sie es durch `LatestVersionArn` das, was Sie für die Funktionsdefinition kopiert haben.
- *subscription-definition-version-arn* Ersetzen Sie es durch `LatestVersionArn` das, was Sie für die Abonnementdefinition kopiert haben.

```
aws greengrass create-group-version \  
--group-id group-id \  
--core-definition-version-arn core-definition-version-arn \  
--resource-definition-version-arn resource-definition-version-arn \  
--connector-definition-version-arn connector-definition-version-arn \  
--function-definition-version-arn function-definition-version-arn \  
--subscription-definition-version-arn subscription-definition-version-arn
```

4. Kopieren Sie den Wert `Version` aus der Ausgabe. Dies ist die ID der Gruppenversion. Mit diesem Wert können Sie die Gruppenversion bereitstellen.

Schritt 9: Erstellen einer Bereitstellung

Stellen Sie die Gruppe auf dem Core-Gerät bereit.

1. Stellen Sie in einem Core-Device-Terminal sicher, dass der AWS IoT Greengrass Daemon läuft.
 - a. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen `root`-Eintrag für `/greengrass/ggc/packages/1.11.6/bin/daemon` enthält, dann wird der Daemon ausgeführt.

- b. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

2. Erstellen einer -Bereitstellung.

- Ersetze es *group-id* durch Id das, was du für die Gruppe kopiert hast.
- *group-version-id* Ersetzen Sie es durch Version das, was Sie für die neue Gruppenversion kopiert haben.

```
aws greengrass create-deployment \  
--deployment-type NewDeployment \  
--group-id group-id \  
--group-version-id group-version-id
```

3. Kopieren Sie die DeploymentId aus der Ausgabe.

4. Abrufen des Bereitstellungsstatus.

- *group-id* Ersetzen Sie es durch Id das, was Sie für die Gruppe kopiert haben.
- *deployment-id* Ersetzen Sie es durch DeploymentId das, was Sie für die Bereitstellung kopiert haben.

```
aws greengrass get-deployment-status \  
--group-id group-id \  
--deployment-id deployment-id
```

Wenn der Status lautet `Success`, war die Bereitstellung erfolgreich. Hilfe zur Problembehebung finden Sie unter [Fehlerbehebung](#).

Testen der Lösung

1. Wählen Sie auf der Startseite der AWS IoT Konsole die Option Test aus.
2. Verwenden Sie für Thema abonnieren die folgenden Werte und wählen Sie dann Abonnieren aus. Der Twilio Notifications Connector veröffentlicht Statusinformationen zu diesem Thema.

Eigenschaft	Wert
Abonnementthema	twilio/message/status
MQTT-Nutzlast-Anzeige	Zeigt Nutzlasten als Zeichenfolgen an

3. Verwenden Sie für In Thema veröffentlichen die folgenden Werte und wählen Sie dann Veröffentlichen, um die Funktion aufzurufen.

Eigenschaft	Wert
Thema	Temperatur/Input
Fehlermeldung	<i>recipient-name</i> Ersetzen Sie es <i>recipient-phone-number</i> durch einen Namen und die Telefonnummer des SMS- Empfängers. Beispiel: +12345000000 <pre>{ "to_name": " <i>recipient-name</i> ", "to_number": " <i>recipient-phone-number</i> ", "temperature": 31 }</pre> Wenn Sie ein Testkonto verwenden, müssen Sie einer Liste verifizierter Telefonnummern hinzufügen, die keine Twilio-Empfänger sind. Weitere Informationen finden Sie unter Verifizieren Sie Ihre persönliche Telefonnummer.

Wenn erfolgreich, erhält der Empfänger die Textnachricht und die Konsole zeigt den Status `success` für die [Ausgangsdaten](#).

Ändern Sie nun den `temperature` in der Eingabemeldung auf **29** und veröffentlichen Sie. Da dieser Wert unter 30 liegt, löst die `TempMonitor` Funktion keine Twilio-Nachricht aus.

Weitere Informationen finden Sie auch unter

- [Integrieren von Services und Protokollen mit Konnektoren](#)
- [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#)
- [the section called “Beginnen Sie mit Konnektoren \(Konsole\)”](#)
- [AWS Secrets Manager Befehle](#) in der AWS CLI Befehlsreferenz
- [AWS Identity and Access Management \(IAM\) -Befehle](#) in der AWS CLI Befehlsreferenz
- [AWS Lambda Befehle](#) in der AWS CLI Befehlsreferenz
- [AWS IoT Greengrass Befehle](#) in der AWS CLI Befehlsreferenz

Greengrass Discovery-API RESTful

Alle Client-Geräte, die mit einem AWS IoT Greengrass Core kommunizieren, müssen Mitglied einer Greengrass-Gruppe sein. Jede Gruppe muss einen Greengrass Core haben. Die Discovery-API ermöglicht es Geräten, Informationen abzurufen, die für die Verbindung mit einem Greengrass-Core erforderlich sind, der sich in derselben Greengrass-Gruppe wie das Client-Gerät befindet. Wenn ein Client-Gerät zum ersten Mal online geht, kann es sich mit dem AWS IoT Greengrass Service verbinden und mithilfe der Discovery-API Folgendes ermitteln:

- Die Gruppe, zu der es gehört Ein Client-Gerät kann Mitglied von bis zu 10 Gruppen sein.
- Die IP-Adresse und der Port für den Greengrass Core in der Gruppe.
- Das CA-Gruppenzertifikat, das zur Authentifizierung des Greengrass Core-Geräts verwendet werden kann.

Note

Client-Geräte können das AWS IoT Gerät auch verwenden SDKs , um Konnektivitätsinformationen für einen Greengrass-Core zu ermitteln. Weitere Informationen finden Sie unter [AWS IoT Geräte-SDK](#).

Senden Sie zur Nutzung dieser API HTTP-Anfragen an den Discovery API-Endpunkt. Beispiel:

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Eine Liste der unterstützten Amazon Web Services Services-Regionen und Endpunkte für die AWS IoT Greengrass Discovery-API finden Sie unter [AWS IoT Greengrass Endpunkte und Kontingente](#) in der. Allgemeine AWS-Referenz Dies ist nur eine Datenebenen-API. Die Endpunkte für die Gruppenverwaltung und den AWS IoT Core Gruppenbetrieb unterscheiden sich von den Discovery-API-Endpunkten.

Anforderung

Die Anfrage enthält die Standard-HTTP-Header und wird an den Greengrass Discovery-Endpunkt gesendet, wie in den folgenden Beispielen gezeigt.

Die Portnummer hängt davon ab, ob der Core für das Senden von HTTPS-Datenverkehr über Port 8443 oder Port 443 konfiguriert ist. Weitere Informationen finden Sie unter [the section called “Verbindungsherstellung auf Port 443 oder über einen Netzwerk-Proxy”](#).

Port 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/thing/thing-name
```

Port 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/thing/thing-name
```

Clients, die eine Verbindung über Port 443 herstellen, müssen die TLS-Erweiterung [Application Layer Protocol Negotiation \(ALPN\)](#) implementieren und `x-amzn-http-ca` sie als in übergeben. `ProtocolName` `ProtocolNameList` Weitere Informationen finden Sie unter [Protokolle](#) im AWS IoT Entwicklerhandbuch.

Note

Diese Beispiele verwenden den Amazon Trust Services (ATS)-Endpunkt, der mit ATS-Stammzertifizierungsstellenzertifikat verwendet wird (empfohlen). Endpunkte müssen dem Typ des Stammzertifizierungsstellenzertifikats entsprechen. Weitere Informationen finden Sie unter [the section called “Dienstendpunkte müssen dem Zertifikatstyp entsprechen”](#).

Antwort

Bei erfolgreicher Anfrage enthält die Antwort die Standard-HTTP-Kopfzeilen sowie den folgenden Code und Text:

```
HTTP 200  
BODY: response document
```

Weitere Informationen finden Sie unter [Beispieldokumente für Discovery-Antwort](#).

Discovery-Autorisierung

Für das Abrufen der Verbindungsinformationen ist eine Richtlinie erforderlich, die es dem Aufrufer erlaubt, die Aktion `greengrass:Discover` durchzuführen. Die einzige akzeptierte Form der Authentifizierung ist die gegenseitige TLS-Authentifizierung mit einem Client-Zertifikat. Im Folgenden finden Sie eine Beispielrichtlinie, die einem Aufrufer das Durchführen dieser Aktion ermöglicht:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "greengrass:Discover",
    "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
  }]
}
```

Beispieldokumente für Discovery-Antwort

Das folgende Dokument zeigt die Antwort für ein Client-Gerät, das Mitglied einer Gruppe mit einem Greengrass-Core-, einem Endpunkt- und einem Gruppen-CA-Zertifikat ist:

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ]
    }
  ],
}
```

```

    "CAs": [
      "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
    ]
  }
]
}

```

Das folgende Dokument zeigt die Antwort für ein Client-Gerät, das Mitglied von zwei Gruppen mit einem Greengrass-Core, mehreren Endpunkten und mehreren Gruppen-CA-Zertifikaten ist:

```

{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-connection-1-description"
            },
            {
              "id": "core-01-connection-id-2",
              "hostAddress": "core-01-address-2",
              "portNumber": core-01-port-2,
              "metadata": "core-01-connection-2-description"
            }
          ]
        }
      ]
    },
    {
      "GGGroupId": "gg-group-02-id",
      "Cores": [
        {

```

```

    "thingArn": "core-02-thing-arn",
    "Connectivity" : [
    {
        "id": "core-02-connection-id",
        "hostAddress": "core-02-address",
        "portNumber": core-02-port,
        "metadata": "core-02-connection-1-description"
    }
    ],
    "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
    ]
  }
}
}

```

Note

Eine Greengrass-Gruppe muss genau einen Greengrass Core definieren. Jede Antwort des AWS IoT Greengrass Dienstes, die eine Liste von Greengrass-Kernen enthält, enthält nur einen Greengrass-Kern.

Wenn Sie cURL installiert haben, können Sie die Discovery-Anfrage testen. Beispiel:

```

$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":[{"thingArn":"arn:aws:iot:us-west-2:123456789012:thing/MyFirstGroup_Core","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"]}]}

```

Sicherheit in AWS IoT Greengrass

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das [Modell der geteilten Verantwortung](#) beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der [AWS](#) . Weitere Informationen zu den Compliance-Programmen, die für gelten AWS IoT Greengrass, finden Sie unter [AWS Services im Umfang nach Compliance-Programmen AWS](#) .
- Sicherheit in der Cloud: Ihr Verantwortungsumfang wird durch den AWS -Service bestimmt, den Sie verwenden. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, die Anforderungen Ihres Unternehmens und die geltenden Gesetze und Vorschriften.

Bei der Nutzung AWS IoT Greengrass sind Sie auch für die Sicherung Ihrer Geräte, Ihrer lokalen Netzwerkverbindung und Ihrer privaten Schlüssel verantwortlich.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung anwenden können AWS IoT Greengrass. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS IoT Greengrass , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer AWS IoT Greengrass Ressourcen unterstützen.

Themen

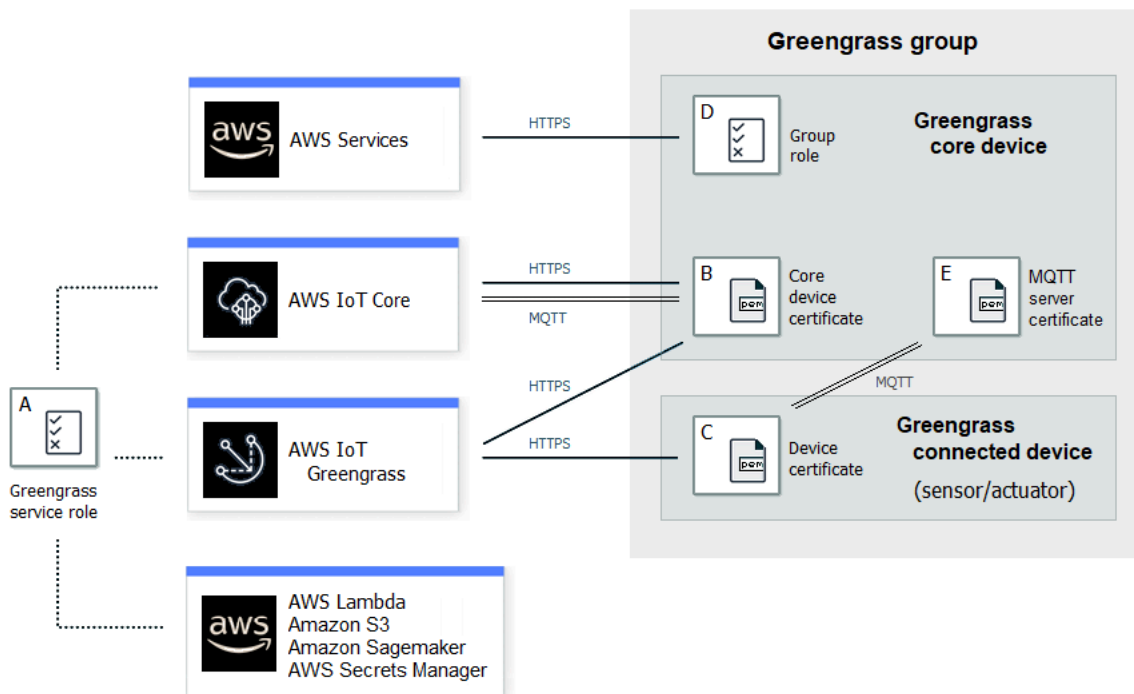
- [Überblick über die AWS IoT Greengrass Sicherheit](#)
- [Datenschutz in AWS IoT Greengrass](#)
- [Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass](#)
- [Identitäts- und Zugriffsmanagement für AWS IoT Greengrass](#)
- [Konformitätsvalidierung für AWS IoT Greengrass](#)
- [Resilienz in AWS IoT Greengrass](#)

- [Infrastruktursicherheit in AWS IoT Greengrass](#)
- [Konfiguration und Schwachstellenanalyse in AWS IoT Greengrass](#)
- [AWS IoT Greengrass und Schnittstellen-VPC-Endpunkte \(AWS PrivateLink\)](#)
- [Bewährte Sicherheitsmethoden für AWS IoT Greengrass](#)

Überblick über die AWS IoT Greengrass Sicherheit

AWS IoT Greengrass verwendet X.509-Zertifikate, AWS IoT Richtlinien sowie IAM-Richtlinien und -Rollen, um die Anwendungen zu sichern, die auf Geräten in Ihrer lokalen Greengrass-Umgebung ausgeführt werden.

Das folgende Diagramm zeigt die Komponenten des AWS IoT Greengrass Sicherheitsmodells:



A – Rolle des Greengrass-Services

Eine vom Kunden erstellte IAM-Rolle, die AWS IoT Greengrass beim Zugriff auf Ihre AWS Ressourcen über AWS IoT Core AWS Lambda, und andere Dienste übernommen wird. AWS Weitere Informationen finden Sie unter [the section called “Greengrass-Servicerolle”](#).

B – Core-Gerätezertifikat

Ein X.509-Zertifikat, das zur Authentifizierung eines Greengrass-Kerns mit und verwendet wird. AWS IoT Core AWS IoT Greengrass Weitere Informationen finden Sie unter [the section called “Geräteauthentifizierung und -autorisierung”](#).

C – Gerätezertifikat

Ein X.509-Zertifikat zur Authentifizierung eines Client-Geräts, das auch als verbundenes Gerät bezeichnet wird, mit und. AWS IoT Core AWS IoT Greengrass Weitere Informationen finden Sie unter [the section called “Geräteauthentifizierung und -autorisierung”](#).

D – Gruppenrolle

Eine vom Kunden erstellte IAM-Rolle, die AWS IoT Greengrass beim Aufrufen von AWS Diensten von einem Greengrass-Core übernommen wird.

Sie verwenden diese Rolle, um Zugriffsberechtigungen anzugeben, die Ihre benutzerdefinierten Lambda-Funktionen und -Konnektoren für den Zugriff auf AWS Dienste wie DynamoDB benötigen. Sie verwenden sie auch, um Stream-Manager-Streams in AWS Dienste AWS IoT Greengrass zu exportieren und in Logs zu schreiben. CloudWatch Weitere Informationen finden Sie unter [the section called “Greengrass-Gruppenrolle.”](#).

Note

AWS IoT Greengrass verwendet nicht die Lambda-Ausführungsrolle, die AWS Lambda für die Cloud-Version einer Lambda-Funktion angegeben ist.

E - MQTT-Serverzertifikat

Das Zertifikat, das für die gegenseitige Authentifizierung mit Transport Layer Security (TLS) zwischen einem Greengrass-Core-Gerät und Client-Geräten in der Greengrass-Gruppe verwendet wird. Das Zertifikat ist durch das Gruppen-CA-Zertifikat signiert, das in der gespeichert ist. AWS Cloud

Geräteverbindung – Workflow

In diesem Abschnitt wird beschrieben, wie Client-Geräte eine Verbindung zum AWS IoT Greengrass Service und zu Greengrass-Core-Geräten herstellen. Client-Geräte sind registrierte AWS IoT Core Geräte, die sich in derselben Greengrass-Gruppe wie das Kerngerät befinden.

- Ein Greengrass-Core-Gerät verwendet sein Gerätezertifikat, seinen privaten Schlüssel und das AWS IoT Core Root-CA-Zertifikat, um eine Verbindung zum AWS IoT Greengrass Dienst herzustellen. Auf dem Core-Gerät gibt das `crypto`-Objekt in der [Konfigurationsdatei](#) den Dateipfad für diese Elemente an.
- Das Greengrass-Kerngerät lädt die Informationen zu Gruppenmitgliedschaften vom AWS IoT Greengrass -Service herunter.
- Wenn eine Bereitstellung auf dem Greengrass-Kerngerät erfolgt, übernimmt der Device Certificate Manager (DCM) die lokale Serverzertifikatverwaltung für das Greengrass-Kerngerät.
- Ein Client-Gerät stellt mithilfe seines Gerätezertifikats, seines privaten Schlüssels und des AWS IoT Core Root-CA-Zertifikats eine Verbindung zum AWS IoT Greengrass Dienst her. Nach dem Herstellen der Verbindung verwendet das Client-Gerät den Greengrass Discovery Service, um die IP-Adresse seines Greengrass-Core-Geräts zu ermitteln. Das Client-Gerät lädt auch das Gruppen-CA-Zertifikat herunter, das für die gegenseitige TLS-Authentifizierung mit dem Greengrass-Core-Gerät verwendet wird.
- Ein Client-Gerät versucht, eine Verbindung zum Greengrass-Core-Gerät herzustellen, wobei es sein Gerätezertifikat und seine Client-ID weitergibt. Wenn die Client-ID mit dem Ding-Namen des Client-Geräts übereinstimmt und das Zertifikat gültig ist (Teil der Greengrass-Gruppe), wird die Verbindung hergestellt. Falls nicht, wird die Verbindungsanfrage beendet.

Die AWS IoT Richtlinie für Client-Geräte muss die `greengrass:Discover` Erlaubnis gewähren, dass Client-Geräte Konnektivitätsinformationen für den Core ermitteln können. Weitere Informationen zur Richtlinienanweisung finden Sie unter [the section called “Discovery-Autorisierung”](#).

AWS IoT Greengrass Sicherheit konfigurieren

So konfigurieren Sie die Sicherheit Ihrer Greengrass-Anwendung:

1. Kreieren Sie AWS IoT Core etwas für Ihr Greengrass-Core-Gerät.
2. Generieren Sie ein Schlüsselpaar und ein Geräte-Zertifikat für Ihr Greengrass-Kerngerät.
3. Erstellen Sie eine [AWS IoT -Richtlinie](#) und fügen Sie sie an das Gerätezertifikat an. Das Zertifikat und die Richtlinie ermöglichen dem Greengrass-Core-Gerät den Zugriff auf AWS IoT Core und die AWS IoT Greengrass Dienste. Weitere Informationen finden Sie unter [Minimale AWS IoT Richtlinie für das Kerngerät](#).

 Note

Die Verwendung von [Ding-Richtlinienvariablen](#) (`iot:Connection.Thing.*`) in der AWS IoT Richtlinie für ein Kerngerät wird nicht unterstützt. Der Core verwendet dasselbe Gerätezertifikat, um [mehrere Verbindungen](#) herzustellen, AWS IoT Core aber die Client-ID in einer Verbindung entspricht möglicherweise nicht exakt dem Namen des Kern-Dings.

4. Erstellen Sie eine [Greengrass-Servicerolle](#). Diese IAM-Rolle autorisiert den AWS IoT Greengrass Zugriff auf Ressourcen von anderen AWS Diensten in Ihrem Namen. Auf diese Weise können AWS IoT Greengrass wichtige Aufgaben ausgeführt werden, z. B. das Abrufen von AWS Lambda Funktionen und das Verwalten von Geräteschatten.

Sie können dieselbe Servicerolle für alle AWS-Region s verwenden, sie muss jedoch an jedem AWS-Region Ort, an dem Sie sie verwenden AWS IoT Greengrass, mit Ihrer AWS-Konto verknüpft sein.

5. (Optional) Erstellen Sie eine [Greengrass-Gruppenrolle](#). Diese IAM-Rolle gewährt Lambda-Funktionen und -Konnektoren, die auf einem Greengrass-Kern ausgeführt werden, die Erlaubnis, Dienste aufzurufen. AWS Beispielsweise benötigt der [Kinesis Firehose-Connector](#) die Erlaubnis, Datensätze in einen Amazon Data Firehose-Lieferstream zu schreiben.

Sie können nur eine Rolle an eine Greengrass-Gruppe anfügen.

6. Erstellen Sie für jedes Gerät, das mit Ihrem Greengrass-Core verbunden ist, ein AWS IoT Core Ding.

 Note

Sie können auch vorhandene AWS IoT Core Dinge und Zertifikate verwenden.

7. Erstellen Sie Gerätezertifikate, Schlüsselpaare und AWS IoT Richtlinien für jedes Gerät, das eine Verbindung zu Ihrem Greengrass Core herstellt.

AWS IoT Greengrass zentrale Sicherheitsprinzipale

Der Greengrass-Kern verwendet die folgenden Sicherheitsprinzipale: AWS IoT Client, lokaler MQTT-Server und Local Secrets Manager. Die Konfiguration für diese Prinzipale ist im `crypto`-Objekt in der

Konfigurationsdatei `config.json` gespeichert. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass Kernkonfigurationsdatei”](#).

Diese Konfiguration beinhaltet den Pfad zum privaten Schlüssel, der von der Prinzipal-Komponente für die Authentifizierung und Verschlüsselung verwendet wird. AWS IoT Greengrass unterstützt zwei Modi der Speicherung privater Schlüssel: hardwarebasiert oder dateisystembasiert (Standard). Weitere Informationen zum Speichern von Schlüsseln auf Hardware-Sicherheitsmodulen finden Sie unter [the section called “Integration von Hardware-Sicherheit”](#).

AWS IoT Client

Der AWS IoT Client (IoT-Client) verwaltet die Kommunikation über das Internet zwischen dem Greengrass-Core und AWS IoT Core. AWS IoT Greengrass verwendet X.509-Zertifikate mit öffentlichen und privaten Schlüsseln für die gegenseitige Authentifizierung beim Aufbau von TLS-Verbindungen für diese Kommunikation. Weitere Informationen finden Sie unter [X.509-Zertifikate und AWS IoT Core](#) im AWS IoT Core -Entwicklerhandbuch.

Der IoT-Client unterstützt RSA- und EC-Zertifikate und -Schlüssel. Der Pfad zu Zertifikaten und privatem Schlüssel sind für den `IoTCertificate`-Prinzipal in `config.json` angegeben.

MQTT-Server

Der lokale MQTT-Server verwaltet die Kommunikation über das lokale Netzwerk zwischen dem Greengrass-Core und den Client-Geräten in der Gruppe. AWS IoT Greengrass verwendet X.509-Zertifikate mit öffentlichen und privaten Schlüsseln für die gegenseitige Authentifizierung beim Aufbau von TLS-Verbindungen für diese Kommunikation.

AWS IoT Greengrass Generiert standardmäßig einen privaten RSA-Schlüssel für Sie. Um den Core so zu konfigurieren, dass er einen anderen privaten Schlüssel verwendet, müssen Sie den Schlüsselpfad für den `MQTTServerCertificate`-Prinzipal in `config.json` angeben. Sie sind dafür verantwortlich, einen vom Kunden bereitgestellten Schlüssel zu rotieren.

Support für privaten Schlüssel

	RSA-Schlüssel	EC-Schlüssel
Schlüsseltyp	Unterstützt	Unterstützt
Hauptparameter	Mindestlänge von 2048 Bit	NIST P-256- oder NIST P-384-Kurve

	RSA-Schlüssel	EC-Schlüssel
Datenträgerformat	PKCS#1, PKCS#8	SECG1, PKCS #8
GGC-Mindestversion	• Verwenden Sie Standard-RSA-Schlüssel: 1.0 • Geben Sie einen RSA-Schlüssel an: 1.7	• Geben Sie einen EC-Schlüssel an: 1.9

Die Konfiguration des privaten Schlüssels bestimmt die zugehörigen Prozesse. Eine Liste der Verschlüsselungs-Suites, die der Greengrass-Core als Server unterstützt, finden Sie unter [the section called “Support für TLS-Verschlüsselungs-Suites”](#).

Wenn kein privater Schlüssel angegeben ist (Standard)

- AWS IoT Greengrass dreht den Schlüssel auf der Grundlage Ihrer Rotationseinstellungen.
- Der Core generiert einen RSA-Schlüssel, der verwendet wird, um das Zertifikat zu generieren.
- Das MQTT-Serverzertifikat verfügt über einen öffentlichen RSA-Schlüssel und eine SHA-256-RSA-Signatur.

Wenn ein privater RSA-Schlüssel angegeben ist (erfordert GGC v1.7 oder höher)

- Sie sind für das Rotieren des Schlüssels verantwortlich.
- Der Core verwendet den angegebenen Schlüssel, um das Zertifikat zu generieren.
- Der RSA-Schlüssel muss eine Mindestlänge von 2048 Bits haben.
- Das MQTT-Serverzertifikat verfügt über einen öffentlichen RSA-Schlüssel und eine SHA-256-RSA-Signatur.

Wenn ein privater EC-Schlüssel angegeben ist (erfordert GGC v1.9 oder höher)

- Sie sind für das Rotieren des Schlüssels verantwortlich.
- Der Core verwendet den angegebenen Schlüssel, um das Zertifikat zu generieren.
- Der private EC-Schlüssel muss eine NIST P-256- oder NIST P-384-Kurve verwenden.
- Das MQTT-Serverzertifikat verfügt über einen öffentlichen EC-Schlüssel und eine SHA-256-RSA-Signatur.

Das vom Core präsentierte MQTT-Serverzertifikat verfügt über eine SHA-256-RSA-Signatur, unabhängig vom Schlüsseltyp. Aus diesem Grund müssen Clients die Validierung

von SHA-256-RSA-Zertifikaten unterstützen, um eine sichere Verbindung mit dem Core herzustellen.

Secrets Manager

Der Local Secrets Manager verwaltet auf sichere Weise lokale Kopien von Geheimnissen, die Sie in erstellen. AWS Secrets Manager Er verwendet einen privaten Schlüssel, um den Datenschlüssel zu sichern, der zum Verschlüsseln der Secrets verwendet wird. Weitere Informationen finden Sie unter [Bereitstellen von Secrets für den Core](#).

Standardmäßig wird der private Schlüssel des IoT-Clients verwendet, aber Sie können einen anderen privaten Schlüssel für den SecretsManager-Prinzipal in `config.json` angeben. Nur der RSA-Schlüsseltyp wird unterstützt. Weitere Informationen finden Sie unter [the section called "Angeben des privaten Schlüssels für die Verschlüsselung von Secrets"](#).

Note

AWS IoT Greengrass Unterstützt derzeit nur den [PKCS #1 v1.5-Padding-Mechanismus](#) für die Verschlüsselung und Entschlüsselung von lokalen Geheimnissen, wenn hardwarebasierte private Schlüssel verwendet werden. Wenn Sie die vom Hersteller bereitgestellten Anweisungen zur manuellen Generierung hardwarebasierter privater Schlüssel befolgen, stellen Sie sicher, dass Sie PKCS #1 v1.5 wählen. AWS IoT Greengrass unterstützt Optimal Asymmetric Encryption Padding (OAEP) nicht.

Support für privaten Schlüssel

	RSA-Schlüssel	EC-Schlüssel
Schlüsseltyp	Unterstützt	Nicht unterstützt
Hauptparameter	Mindestlänge von 2048 Bit	Nicht zutreffend
Datenträgerformat	PKCS#1, PKCS#8	Nicht zutreffend
Minimale GGC-Version	1,7	Nicht zutreffend

Verwaltete Abonnements im MQTT Messaging-Workflow

AWS IoT Greengrass verwendet eine Abonnementtabelle, um zu definieren, wie MQTT-Nachrichten zwischen Client-Geräten, Funktionen und Konnektoren in einer Greengrass-Gruppe und mit AWS IoT Core oder dem lokalen Shadow-Service ausgetauscht werden können. Jedes Abonnement spezifiziert eine Quelle, ein Ziel und ein MQTT-Thema (oder einen Betreff), über das Nachrichten gesendet oder empfangen werden. AWS IoT Greengrass erlaubt das Senden von Nachrichten von einer Quelle an ein Ziel nur, wenn ein entsprechendes Abonnement definiert ist.

Ein Abonnement definiert den Nachrichtenfluss nur in eine Richtung, von der Quelle zum Ziel. Um den bidirektionalen Nachrichtenaustausch zu unterstützen, müssen Sie zwei Abonnements anlegen, eines für jede Richtung.

Support für TLS-Verschlüsselungs-Suites

AWS IoT Greengrass verwendet das AWS IoT Core Transportsicherheitsmodell, um die Kommunikation mit der Cloud mithilfe von [TLS-Verschlüsselungssammlungen](#) zu verschlüsseln. Darüber hinaus werden AWS IoT Greengrass Daten im Ruhezustand (in der Cloud) verschlüsselt. Weitere Informationen zur AWS IoT Core Transportsicherheit und den unterstützten Cipher Suites finden Sie unter [Transport Security](#) im AWS IoT Core Developer Guide.

Unterstützte Verschlüsselungs-Suites für lokale Netzwerkkommunikation

Im Gegensatz dazu AWS IoT Core unterstützt der AWS IoT Greengrass Core die folgenden TLS-Verschlüsselungssammlungen für lokale Netzwerke für Algorithmen zur Zertifikatssignierung. Alle diese Verschlüsselungs-Suites werden unterstützt, wenn private Schlüssel im Dateisystem gespeichert werden. Eine Teilmenge wird unterstützt, wenn der Core so konfiguriert ist, dass Hardware-Sicherheitsmodule (HSM) verwendet werden. Weitere Informationen erhalten Sie unter [the section called "Sicherheitsprinzipale"](#) und [the section called "Integration von Hardware-Sicherheit"](#). Die Tabelle enthält auch die Mindestversion der AWS IoT Greengrass Core-Software, die für die Unterstützung erforderlich ist.

	Verschlüsselungsverfahren	HSM-Support	GGC-Mindestversion
TLSv12.	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Unterstützt	1,0

Verschlüsselungsverfahren	HSM-Support	GGC-Mindestversion
TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Unterstützt	1,0
TLS_ECDHE _RSA_WITH _AES_256_GCM_ SHA384	Unterstützt	1,0
TLS_RSA_W ITH_AES_1 28_CBC_SHA	Nicht unterstützt	1,0
TLS_RSA_M IT_AES_128_GCM_ SHA256	Nicht unterstützt	1,0
TLS_RSA_W ITH_AES_2 56_CBC_SHA	Nicht unterstützt	1,0
TLS_RSA_M IT_AES_256_GCM_ SHA384	Nicht unterstützt	1,0
TLS_ECDHE _ECDSA_MI T_AES_128_GCM_ SHA256	Unterstützt	1.9
TLS_ECDHE _ECDSA_WI TH_AES_256_GCM_ SHA384	Unterstützt	1.9

	Verschlüsselungsverfahren	HSM-Support	GGC-Mindestversion
TLSv11.	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Unterstützt	1,0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Unterstützt	1,0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Nicht unterstützt	1,0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Nicht unterstützt	1,0
TLSv10.	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Unterstützt	1,0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Unterstützt	1,0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Nicht unterstützt	1,0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Nicht unterstützt	1,0

Datenschutz in AWS IoT Greengrass

Das [Modell der AWS gemeinsamen Verantwortung](#) und geteilter Verantwortung gilt für den Datenschutz in AWS IoT Greengrass. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter [Häufig gestellte Fragen zum Datenschutz](#). Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag [AWS -Modell der geteilten Verantwortung und in der DSGVO](#) im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Verwenden Sie SSL/TLS, um mit Ressourcen zu kommunizieren. AWS Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein. AWS CloudTrail Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter [Arbeiten mit CloudTrail Pfaden](#) im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-3](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der AWS IoT Greengrass API oder auf andere AWS-Services Weise arbeiten oder diese verwenden. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet

werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Weitere Hinweise zum Schutz vertraulicher Informationen finden Sie unter [the section called “Keine Protokollierung sensibler Informationen”](#). AWS IoT Greengrass

Weitere Informationen zum Datenschutz enthält der Blog-Beitrag [AWS Shared Responsibility Model and GDPR](#) im AWS -Sicherheitsblog.

Themen

- [Datenverschlüsselung](#)
- [Integration von Hardware-Sicherheit](#)

Datenverschlüsselung

AWS IoT Greengrass verwendet Verschlüsselung, um Daten während der Übertragung (über das Internet oder ein lokales Netzwerk) und im Ruhezustand (gespeichert im AWS Cloud) zu schützen.

Geräte in einer AWS IoT Greengrass Umgebung sammeln häufig Daten, die zur weiteren Verarbeitung an AWS Dienste gesendet werden. Weitere Informationen zur Datenverschlüsselung bei anderen AWS Diensten finden Sie in der Sicherheitsdokumentation für diesen Dienst.

Themen

- [Verschlüsselung während der Übertragung](#)
- [Verschlüsselung im Ruhezustand](#)
- [Schlüsselverwaltung für das Greengrass Core-Gerät](#)

Verschlüsselung während der Übertragung

AWS IoT Greengrass hat drei Kommunikationsarten, bei denen Daten übertragen werden:

- [the section called “Daten in Übertragung über das Internet”](#). Die Kommunikation zwischen einem Greengrass-Kern und AWS IoT Greengrass über das Internet ist verschlüsselt.
- [the section called “Daten in Übertragung über das lokale Netzwerk”](#). Die Kommunikation zwischen einem Greengrass-Core und Client-Geräten über ein lokales Netzwerk ist verschlüsselt.

- [the section called “Daten auf dem Core-Gerät”](#). Die Kommunikation zwischen Komponenten auf dem Greengrass-Core-Gerät ist nicht verschlüsselt.

Daten in Übertragung über das Internet

AWS IoT Greengrass verwendet Transport Layer Security (TLS), um die gesamte Kommunikation über das Internet zu verschlüsseln. Alle an die AWS Cloud gesendeten Daten werden über eine TLS-Verbindung unter Verwendung der Protokolle MQTT oder HTTPS gesendet, sodass sie standardmäßig sicher sind. AWS IoT Greengrass verwendet das AWS IoT Transportsicherheitsmodell. Weitere Informationen finden Sie unter [Transportsicherheit](#) im AWS IoT Core -Entwicklerhandbuch.

Daten in Übertragung über das lokale Netzwerk

AWS IoT Greengrass verwendet TLS, um die gesamte Kommunikation über das lokale Netzwerk zwischen dem Greengrass-Core und den Client-Geräten zu verschlüsseln. Weitere Informationen finden Sie unter [Unterstützte Verschlüsselungssammlungen für die lokale Netzwerkkommunikation](#).

Sie sind für den Schutz des lokalen Netzwerks und privater Schlüssel verantwortlich.

Für Greengrass Kerngeräte sind Sie für Folgendes verantwortlich:

- Ständige Aktualität des Kernels mit den neuesten Sicherheits-Patches.
- Ständige Aktualität der Systembibliotheken durch neueste Sicherheits-Patches.
- Schutz privater Schlüssel. Weitere Informationen finden Sie unter [the section called “Schlüsselverwaltung”](#).

Bei Client-Geräten liegt es in Ihrer Verantwortung:

- Ständige Aktualität des TLS-Stacks.
- Schutz privater Schlüssel.

Daten auf dem Core-Gerät

AWS IoT Greengrass verschlüsselt keine lokal auf dem Greengrass-Core-Gerät ausgetauschten Daten, da die Daten das Gerät nicht verlassen. Dazu gehört die Kommunikation zwischen benutzerdefinierten Lambda-Funktionen, Konnektoren, dem AWS IoT Greengrass Core SDK und Systemkomponenten wie dem Stream Manager.

Verschlüsselung im Ruhezustand

AWS IoT Greengrass speichert Ihre Daten:

- [the section called “Daten im Ruhezustand im AWS Cloud”](#). Diese Daten sind verschlüsselt.
- [the section called “Daten im Ruhezustand auf dem Greengrass-Kern”](#). Diese Daten sind nicht verschlüsselt (außer lokalen Kopien Ihrer Geheimnisse).

Daten im Ruhezustand im AWS Cloud

AWS IoT Greengrass verschlüsselt Kundendaten, die in der AWS Cloud gespeichert sind. Diese Daten werden durch AWS KMS Schlüssel geschützt, die von AWS IoT Greengrass verwaltet werden.

Daten im Ruhezustand auf dem Greengrass-Kern

AWS IoT Greengrass stützt sich auf Unix-Dateiberechtigungen und vollständige Festplattenverschlüsselung (falls aktiviert), um Daten zu schützen, die sich auf dem Kern befinden. Sie sind für den Schutz des Dateisystems und des Geräts verantwortlich.

Verschlüsselt AWS IoT Greengrass jedoch lokale Kopien Ihrer abgerufenen Geheimnisse. AWS Secrets Manager Weitere Informationen finden Sie unter [the section called “Verschlüsselung von Secrets”](#).

Schlüsselverwaltung für das Greengrass Core-Gerät

Es liegt in der Verantwortung des Kunden, die sichere Speicherung von kryptografischen (öffentlichen und privaten) Schlüsseln auf dem Greengrass-Kerngerät zu gewährleisten. AWS IoT Greengrass verwendet öffentliche und private Schlüssel für die folgenden Szenarien:

- Der IoT-Clientschlüssel wird zusammen mit dem IoT-Zertifikat verwendet, um den Transport Layer Security (TLS) Handshake zu authentifizieren, wenn ein Greengrass-Kern eine Verbindung zu AWS IoT Core herstellt. Weitere Informationen finden Sie unter [the section called “Geräteauthentifizierung und -autorisierung”](#).

Note

Der Schlüssel und das Zertifikat werden auch als der private Core-Schlüssel und das Core-Gerätezertifikat bezeichnet.

- Der MQTT-Serverschlüssel wird als MQTT-Serverzertifikat verwendet, um TLS-Verbindungen zwischen Core- und Client-Geräten zu authentifizieren. Weitere Informationen finden Sie unter [the section called “Geräteauthentifizierung und -autorisierung”](#).
- Der lokale Secrets Manager verwendet auch den IoT-Clientschlüssel, um den Datenschlüssel zu schützen, der zum Verschlüsseln lokaler Geheimnisse verwendet wird. Sie können jedoch Ihren eigenen privaten Schlüssel angeben. Weitere Informationen finden Sie unter [the section called “Verschlüsselung von Secrets”](#).

Ein Greengrass-Kern unterstützt den privaten Schlüsselspeicher mithilfe von Dateisystemberechtigungen, [Hardware Sicherheitsmodulen](#) oder beiden. Wenn Sie auf dem Dateisystem basierende private Schlüssel verwenden, sind Sie für die sichere Speicherung auf dem Kerngerät verantwortlich.

Auf einem Greengrass-Kern wird der Speicherort Ihrer privaten Schlüssel im `crypto`-Abschnitt der `config.json`-Datei angegeben. Wenn Sie den Kern so konfigurieren, dass er einen vom Kunden bereitgestellten Schlüssel für das MQTT-Serverzertifikat verwendet, liegt es in Ihrer Verantwortung, den Schlüssel zu rotieren. Weitere Informationen finden Sie unter [the section called “Sicherheitsprinzipale”](#).

Bei Client-Geräten liegt es in Ihrer Verantwortung, den TLS-Stack auf dem neuesten Stand zu halten und private Schlüssel zu schützen. Private Schlüssel werden zusammen mit Gerätezertifikaten verwendet, um TLS-Verbindungen mit dem AWS IoT Greengrass Dienst zu authentifizieren.

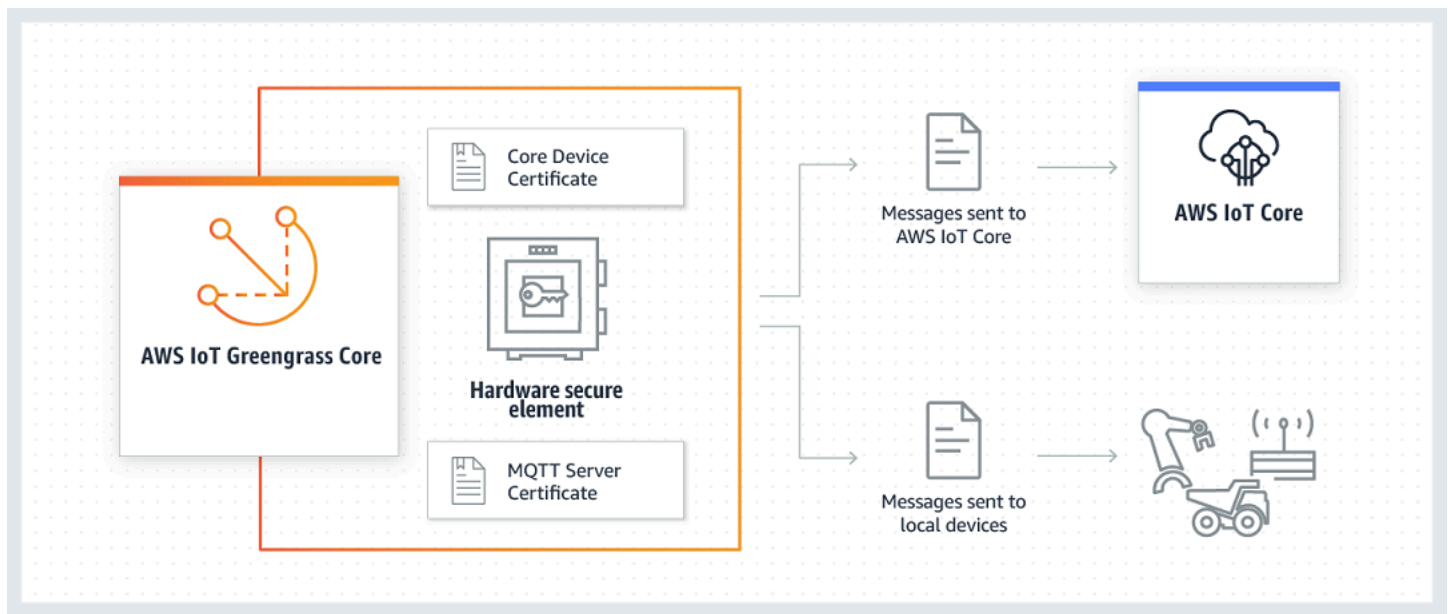
Integration von Hardware-Sicherheit

Diese Funktion ist für AWS IoT Greengrass Core v1.7 und höher verfügbar.

AWS IoT Greengrass unterstützt die Verwendung von Hardware-Sicherheitsmodulen (HSM) über die [PKCS #11 -Schnittstelle](#) für die sichere Speicherung und Übertragung von privaten Schlüsseln. Dadurch wird verhindert, dass Schlüssel in der Software freigelegt oder dupliziert werden. Private Schlüssel können sicher auf Hardwaremodulen wie HSMs Trusted Platform Modules (TPM) oder anderen kryptografischen Elementen gespeichert werden.

Suchen Sie im [AWS Partner Gerätecatalog](#) nach Geräten, die für diese Funktion qualifiziert sind.

Das folgende Diagramm zeigt die Hardware-Sicherheitsarchitektur für einen AWS IoT Greengrass Kern.



AWS IoT Greengrass Verwendet bei einer Standardinstallation zwei private Schlüssel. Ein Schlüssel wird von der AWS IoT Client-Komponente (IoT-Client) während des Transport Layer Security (TLS) - Handshakes verwendet, wenn ein Greengrass-Core eine Verbindung herstellt. AWS IoT Core(Dieser Schlüssel wird auch als privater Kernschlüssel bezeichnet.) Der andere Schlüssel wird vom lokalen MQTT-Server verwendet, wodurch Greengrass-Geräte mit dem Greengrass Core kommunizieren können. Wenn Sie die Hardware-Sicherheit für beide Komponenten verwenden möchten, können Sie einen gemeinsamen privaten Schlüssel oder separate private Schlüssel verwenden. Weitere Informationen finden Sie unter [the section called “Bereitstellungspraktiken”](#).

Note

Bei einer Standardinstallation verwendet der lokale Secrets-Manager auch den IoT-Client-Schlüssel für seinen Verschlüsselungsprozess, aber Sie können auch Ihren eigenen privaten Schlüssel verwenden. Es muss ein RSA-Schlüssel mit einer minimalen Länge von 2048 Bit sein. Weitere Informationen finden Sie unter [the section called “Angaben des privaten Schlüssels für die Verschlüsselung von Secrets”](#).

Voraussetzungen

Bevor Sie die Hardwaresicherheit für einen Greengrass Core konfigurieren können, müssen Sie Folgendes beachten:

- Ein Hardwaresicherheitsmodul (HSM), das Ihre private Schlüsselkonfiguration für den IoT-Client, den lokalen MQTT-Server und lokale Secrets-Manager-Komponenten unterstützt. Die Konfiguration kann einen, zwei oder drei hardwarebasierte private Schlüssel beinhalten, je nachdem, ob Sie die Komponenten für die Freigabe von Schlüsseln konfigurieren. Weitere Informationen zur Unterstützung von privaten Schlüsseln finden Sie unter [the section called “Sicherheitsprinzipale”](#).
- Für RSA-Schlüssel: Eine RSA-Schlüsselgröße von 2048 (oder größer) und ein [PKCS#1 v1.5](#)-Signaturschema.
- Für EC-Schlüssel: Eine NIST P-256- oder NIST P-384-Kurve.

 Note

Suchen Sie im [AWS Partner Gerätecatalog](#) nach Geräten, die für diese Funktion qualifiziert sind.

- Eine PKCS#11-Anbieterbibliothek, die zur Laufzeit (mit libdl) geladen werden kann und [PKCS#11](#)-Funktionen bereitstellt.
- Das Hardwaremodul muss nach Slot-Label auflösbar sein, wie in der PKCS#11-Spezifikation definiert.
- Der private Schlüssel muss mit Hilfe der vom Hersteller bereitgestellten Provisionierungswerkzeuge generiert und auf das HSM geladen werden.
- Der private Schlüssel muss durch ein Objektlabel auflösbar sein.
- Das Core-Gerätezertifikat. Dies ist ein IoT-Client-Zertifikat, das dem privaten Schlüssel entspricht.
- Wenn Sie den Greengrass OTA Update Agent verwenden, muss die [OpenSSL libp11 PKCS #11](#)-Wrapper-Bibliothek installiert sein. Weitere Informationen finden Sie unter [the section called “Konfigurieren von OTA-Updates”](#).

Stellen Sie außerdem sicher, dass die folgenden Bedingungen erfüllt sind:

- Die IoT-Client-Zertifikate, die dem privaten Schlüssel zugeordnet sind, sind registriert AWS IoT und aktiviert. Sie können dies in der AWS IoT Konsole unter Verwalten überprüfen, Alle Geräte erweitern, Dinge auswählen und dann die Registerkarte Zertifikate für den Kern auswählen.
- Die AWS IoT Greengrass Core-Software v1.7 oder höher ist auf dem Core-Gerät installiert, wie in [Modul 2](#) des Tutorials Erste Schritte beschrieben. Version 1.9 oder höher ist erforderlich, um einen EC-Schlüssel für den MQTT-Server zu verwenden.

- Die Zertifikate sind dem Greengrass Core angefügt. Sie können dies auf der Seite „Verwalten“ für den Kern in der AWS IoT Konsole überprüfen.

Note

Unterstützt derzeit AWS IoT Greengrass nicht das Laden des CA-Zertifikats oder des IoT-Client-Zertifikats direkt aus dem HSM. Die Zertifikate müssen als Klartextdateien an einem Ort im Dateisystem geladen werden, der von Greengrass gelesen werden kann.

Hardwaresicherheitskonfiguration für einen Kern AWS IoT Greengrass

Die Hardware-Sicherheit wird in der Greengrass-Konfigurationsdatei konfiguriert. Dies ist die Datei [config.json](#), die sich im Verzeichnis `/greengrass-root/config` befindet.

Note

Um den Prozess der Einrichtung einer HSM-Konfiguration mit einer reinen Software-Implementierung zu durchlaufen, siehe [the section called “Modul 7: Simulation der Hardware-Sicherheitsintegration”](#).

Important

Die simulierte Konfiguration im Beispiel bietet keine Sicherheitsvorteile. Das Beispiel soll Ihnen ermöglichen, sich über die PKCS#11-Spezifikation zu informieren und erste Tests Ihrer Software durchzuführen, wenn Sie planen, in Zukunft ein hardwarebasiertes HSM einzusetzen.

Um die Hardwaresicherheit in zu konfigurieren AWS IoT Greengrass, bearbeiten Sie das `crypto` Objekt in `config.json`.

Bei der Verwendung von Hardware-Sicherheit wird das Objekt `crypto` eingesetzt, um Pfade zu Zertifikaten, privaten Schlüsseln und Assets für die PKCS#11-Anbieterbibliothek im Core anzugeben, wie im folgenden Beispiel gezeigt.

```
"crypto": {  
  "PKCS11" : {
```

```

"OpenSSLEngine" : "/path-to-p11-openssl-engine",
"P11Provider" : "/path-to-pkcs11-provider-so",
"slotLabel" : "crypto-token-name",
"slotUserPin" : "crypto-token-user-pin"
},
"principals" : {
  "IoTCertificate" : {
    "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
    "certificatePath" : "file:///path-to-core-device-certificate"
  },
  "MQTTServerCertificate" : {
    "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
  },
  "SecretsManager" : {
    "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
  }
},
"caPath" : "file:///path-to-root-ca"

```

Das crypto-Objekt enthält die folgenden Eigenschaften:

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
 Note Stellen Sie sicher, dass Ihre Endpunkte Ihrem Zertifikatstyp entsprechen.		
PKCS11		
OpenSSLEngine	Optional. Der absolute Pfad zur OpenSSL-Engine . so-Datei, um die PKCS#11-U	Muss ein Pfad zu einer Datei im Dateisystem sein.

Feld	Description	Hinweise
	Unterstützung unter OpenSSL zu aktivieren.	Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates” .
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	

Feld	Description	Hinweise
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate.privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .

 Note

Stellen Sie sicher, dass Ihre [Endpunkte Ihrem Zertifikatstyp entsprechen](#).

PKCS11

Feld	Description	Hinweise
OpenSSLEngine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren.	Muss ein Pfad zu einer Datei im Dateisystem sein. Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates” .
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	

Feld	Description	Hinweise
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate.privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Feld	Description	Hinweise
caPath	Der absolute Pfad zur AWS IoT Stammzertifizierungsstelle.	Muss ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> .

 Note

Stellen Sie sicher, dass Ihre [Endpunkte Ihrem Zertifikatstyp entsprechen](#).

PKCS11

Feld	Description	Hinweise
OpenSSLEngine	Optional. Der absolute Pfad zur OpenSSL-Engine .so-Datei, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren.	Muss ein Pfad zu einer Datei im Dateisystem sein. Diese Eigenschaft ist erforderlich, wenn Sie den Greengrass OTA Update Agent mit Hardwaresicherheit verwenden. Weitere Informationen finden Sie unter the section called “Konfigurieren von OTA-Updates” .
P11Provider	Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.	Muss ein Pfad zu einer Datei im Dateisystem sein.
slotLabel	Das Slot-Label, das zur Identifizierung des Hardwaremoduls verwendet wird.	Muss den PKCS#11 Label-Spezifikationen entsprechen.
slotUserPin	Die Benutzer-PIN, die zur Authentifizierung des Greengrass-Kerns gegenüber dem Modul verwendet wird.	Muss über ausreichende Berechtigungen verfügen, um C_Sign mit den konfigurierten privaten Schlüsseln auszuführen.
principals		
IoTCertificate	Das Zertifikat und der private Schlüssel, die Core verwendet, um Anfragen AWS IoT zu stellen.	

Feld	Description	Hinweise
IoTCertificate .privateKeyPath	Der Pfad zum privaten Core-Schlüssel.	Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt.
IoTCertificate .certificatePath	Der absolute Pfad zum Core-Gerätezertifikat.	Muss ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> .
MQTTServerCertificate	Optional. Der private Schlüssel, den der Kern in Kombination mit dem Zertifikat verwendet, um als MQTT-Server oder Gateway zu fungieren.	

Feld	Description	Hinweise
MQTTServerCertificate.privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen MQTT-Servers.	Verwenden Sie diesen Wert, um einen eigenen privaten Schlüssel für den lokalen MQTT-Server anzugeben. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <i>file:///absolute/path/to/file</i> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Wenn diese Eigenschaft weggelassen wird, wird der Schlüssel auf der Grundlage Ihrer AWS IoT Greengrass Rotationseinstellungen rotiert. Sofern angegeben, ist der Kunde für das Rotieren des Schlüssels verantwortlich.
SecretsManager	Der private Schlüssel, der den für die Verschlüsselung verwendeten Datenschlüssel schützt. Weitere Informationen finden Sie unter Bereitstellen von Secrets für den Core .	

Feld	Description	Hinweise
SecretsManager .privateKeyPath	Der Pfad zum privaten Schlüssel des lokalen Secrets Managers.	Nur ein RSA-Schlüssel wird unterstützt. Muss für den Dateisystemspeicher ein Datei-URI im folgenden Format sein: <code>file:///absolute/path/to/file</code> . Muss für einen HSM-Speicher ein RFC 7512 PKCS#11-Pfad sein, der eine Objektbezeichnung angibt. Der private Schlüssel muss erstellt werden, indem Sie den PKCS#1 v1.5-Padding-Mechanismus verwenden.

Bereitstellungspraktiken für AWS IoT Greengrass die Hardwaresicherheit

Im Folgenden werden die sicherheits- und leistungsbezogenen Bereitstellungspraktiken beschrieben.

Sicherheit

- Generieren Sie private Schlüssel direkt auf dem HSM mit Hilfe des internen Hardware-Zufallszahlengenerators.

Note

Wenn Sie private Schlüssel für die Verwendung mit dieser Funktion konfigurieren (indem Sie die Anweisungen des Hardwareanbieters befolgen), beachten Sie, dass AWS IoT Greengrass derzeit nur der Padding-Mechanismus der Version PKCS1 1.5 für die Verschlüsselung und Entschlüsselung von lokalen Geheimnissen unterstützt wird.
AWS IoT Greengrass unterstützt Optimal Asymmetric Encryption Padding (OAEP) nicht.

- Konfigurieren Sie private Schlüssel, um den Export zu verhindern.

- Verwenden Sie das vom Hardwareanbieter bereitgestellte Bereitstellungstool, um mithilfe des hardwaregeschützten privaten Schlüssels eine Zertifikatsignieranforderung (Certificate Signing Request, CSR) zu generieren, und verwenden Sie dann die Konsole, um ein Client-Zertifikat zu generieren. AWS IoT

 Note

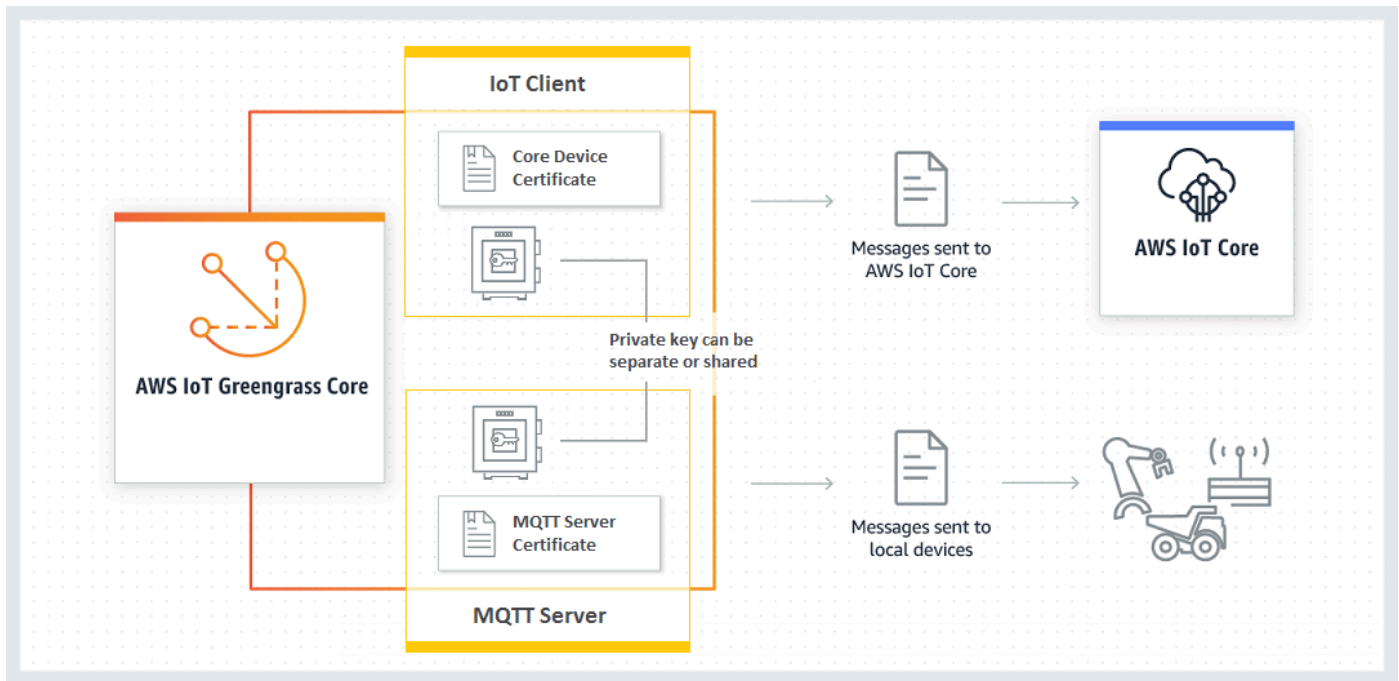
Die Praxis des Rotierens von Schlüsseln gilt nicht, wenn private Schlüssel auf einem HSM erzeugt werden.

Leistung

Das folgende Diagramm zeigt die IoT-Client-Komponente und den lokalen MQTT-Server auf dem AWS IoT Greengrass Core. Wenn Sie eine HSM-Konfiguration für beide Komponenten verwenden möchten, können Sie den gleichen privaten Schlüssel oder separate private Schlüssel verwenden. Wenn Sie separate Schlüssel verwenden, müssen diese im gleichen Slot gespeichert sein.

 Note

AWS IoT Greengrass schränkt die Anzahl der Schlüssel, die Sie auf dem HSM speichern, nicht ein, sodass Sie private Schlüssel für die Komponenten IoT-Client, MQTT-Server und Secrets Manager speichern können. Einige HSM-Anbieter können jedoch die Anzahl der Schlüssel, die Sie in einem Slot speichern können, begrenzen.



Im Allgemeinen wird der IoT-Clientschlüssel nicht sehr häufig verwendet, da die AWS IoT Greengrass Core-Software langlebige Verbindungen zur Cloud unterhält. Der MQTT-Server-Schlüssel wird jedoch jedes Mal verwendet, wenn sich ein Greengrass-Gerät mit dem Kern verbindet. Diese Interaktionen wirken sich direkt auf die Leistung aus.

Wenn der MQTT-Server-Schlüssel auf dem HSM gespeichert ist, hängt die Geschwindigkeit, mit der sich Geräte verbinden können, von der Anzahl der RSA-Signaturvorgänge pro Sekunde ab, die das HSM ausführen kann. Wenn das HSM beispielsweise 300 Millisekunden benötigt, um eine RSASSA-PKCS1-v1.5-Signatur auf einem privaten RSA-2048-Schlüssel durchzuführen, können nur drei Geräte pro Sekunde eine Verbindung zum Greengrass-Core herstellen. Nach dem Herstellen der Verbindungen wird das HSM nicht mehr verwendet, und es gelten die standardmäßigen [Kontingente für AWS IoT Greengrass](#).

Um Leistungsengpässe zu vermeiden, können Sie den privaten Schlüssel für den MQTT-Server auf dem Dateisystem statt auf dem HSM speichern. Bei dieser Konfiguration verhält sich der MQTT-Server so, als ob die Hardware-Sicherheit nicht aktiviert wäre.

AWS IoT Greengrass unterstützt mehrere Schlüsselspeicherkonfigurationen für die IoT-Client- und MQTT-Serverkomponenten, sodass Sie Ihre Sicherheits- und Leistungsanforderungen optimieren können. Die folgende Tabelle enthält Beispielskonfigurationen.

Konfiguration	IoT-Schlüssel	MQTT-Schlüssel	Leistung
Gemeinsamer HSM-Schlüssel	HSM: Schlüssel A	HSM: Schlüssel A	Begrenzt durch das HSM oder die CPU
HSM Separate Schlüssel	HSM: Schlüssel A	HSM: Schlüssel B	Begrenzt durch das HSM oder die CPU
HSM nur für IoT	HSM: Schlüssel A	Dateisystem: Schlüssel B	Begrenzt durch die CPU
Veraltet	Dateisystem: Schlüssel A	Dateisystem: Schlüssel B	Begrenzt durch die CPU

Um den Greengrass-Kern so zu konfigurieren, dass er dateisystembasierte Schlüssel für den MQTT-Server verwendet, lassen Sie den `principals.MQTTServerCertificate` Abschnitt von `weg config.json` (oder geben Sie einen dateibasierten Pfad zum Schlüssel an, wenn Sie nicht den von generierten Standardschlüssel verwenden). AWS IoT Greengrass Das resultierende Objekt `crypto` sieht folgendermaßen aus:

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

Unterstützte Verschlüsselungs-Suiten für die Hardware-Sicherheitsintegration

AWS IoT Greengrass unterstützt eine Reihe von Cipher Suites, wenn der Kern für Hardware-Sicherheit konfiguriert ist. Dies ist eine Teilmenge der Verschlüsselungs-Suiten, die unterstützt werden, wenn der Kern für die Verwendung dateibasierter Sicherheit konfiguriert ist. Weitere Informationen finden Sie unter [the section called “Support für TLS-Verschlüsselungs-Suites”](#).

Note

Wenn Sie sich von Greengrass-Geräten über das lokale Netzwerk mit dem Greengrass-Core verbinden, stellen Sie sicher, dass Sie eine der unterstützten Verschlüsselungs-Suiten verwenden, um die TLS-Verbindung herzustellen.

Konfigurieren Sie die Unterstützung für Updates over-the-air

Um over-the-air (OTA-) Updates der AWS IoT Greengrass Core-Software bei Verwendung von Hardwaresicherheit zu aktivieren, müssen Sie die OpenSC libp11 [PKCS #11 -Wrapper-Bibliothek](#) installieren und die Greengrass-Konfigurationsdatei bearbeiten. Weitere Informationen zu OTA-Updates finden Sie unter [OTA-Updates der AWS IoT Greengrass Core-Software](#).

1. Stoppen Sie den Greengrass-Daemon.

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

Note

greengrass-root steht für den Pfad, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

2. Installieren der OpenSSL-Engine. OpenSSL 1.0 oder 1.1 werden unterstützt.

```
sudo apt-get install libengine-pkcs11-openssl
```

3. Suchen Sie den Pfad zur OpenSSL-Engine (`libpkcs11.so`) auf Ihrem System:
 - a. Holen Sie sich die Liste der installierten Pakete für die Bibliothek.

```
sudo dpkg -L libengine-pkcs11-openssl
```

Die Datei `libpkcs11.so` befindet sich im Verzeichnis `engines`.

- b. Kopieren Sie den vollständigen Pfad in die Datei (z. B. `/usr/lib/ssl/engines/libpkcs11.so`).
4. Öffnen Sie die Greengrass-Konfigurationsdatei. Dies ist die Datei [`config.json`](#) im Verzeichnis `/greengrass-root/config`.
5. Geben Sie für die Eigenschaft `OpenSSLEngine` den Pfad zur Datei `libpkcs11.so` ein.

```
{
  "crypto": {
    "caPath" : "file:///path-to-root-ca",
    "PKCS11" : {
      "OpenSSLEngine" : "/path-to-p11-openssl-engine",
      "P11Provider" : "/path-to-pkcs11-provider-so",
      "slotLabel" : "crypto-token-name",
      "slotUserPin" : "crypto-token-user-pin"
    },
    ...
  }
  ...
}
```

Note

Wenn die Eigenschaft `OpenSSLEngine` im Objekt `PKCS11` nicht vorhanden ist, dann fügen Sie sie hinzu.

6. Halten Sie den Greengrass-Daemon an.

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

Abwärtskompatibilität mit früheren Versionen der AWS IoT Greengrass Kernsoftware

Die AWS IoT Greengrass Core-Software mit Unterstützung für Hardwaresicherheit ist vollständig abwärtskompatibel mit `config.json` Dateien, die für Version 1.6 und frühere Versionen generiert

wurden. Wenn das `crypto` Objekt nicht in der `config.json` Konfigurationsdatei vorhanden ist, werden die dateibasierten Eigenschaften `coreThing.certPath`, `coreThing.keyPath` und `coreThing.caPath` von AWS IoT Greengrass verwendet. Diese Abwärtskompatibilität gilt für Greengrass OTA-Updates, die eine dateibasierte Konfiguration, die in `config.json` angegeben ist, nicht überschreiben.

Hardware ohne PKCS#11-Unterstützung

Die PKCS#11-Bibliothek wird typischerweise vom Hardwarehersteller bereitgestellt oder ist Open Source. Bei standardkonformer Hardware (wie TPM1.2) könnte es beispielsweise möglich sein, vorhandene Open-Source-Software zu verwenden. Wenn Ihre Hardware jedoch nicht über eine entsprechende PKCS #11 -Bibliotheksimplementierung verfügt oder wenn Sie einen benutzerdefinierten PKCS #11 -Anbieter schreiben möchten, sollten Sie sich mit Fragen zur Integration an Ihren AWS Enterprise Support-Mitarbeiter wenden.

Weitere Informationen finden Sie auch unter

- PKCS #11 Kryptographische Token-Schnittstelle, Bedienungsanleitung Version 2.40. Herausgegeben von John Leiseboer und Robert Griffin. 16. November 2014. Anmerkung 02 des OASIS-Ausschusses. <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html>. Letzte Version: <http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html>.
- [RFC 7512](#)
- [PKCS #1: RSA Encryption Version 1.5](#)

Geräteauthentifizierung und Autorisierung für AWS IoT Greengrass

Geräte in AWS IoT Greengrass Umgebungen verwenden X.509-Zertifikate für die Authentifizierung und AWS IoT Richtlinien für die Autorisierung. Zertifikate und Richtlinien ermöglichen es Geräten, sich sicher miteinander zu verbinden AWS IoT Core, und AWS IoT Greengrass.

X.509-Zertifikate sind digitale Zertifikate, die den X.509-Standard für eine Public-Key-Infrastruktur nutzen, um einen öffentlichen Schlüssel mit der Identität in einem Zertifikat zu verknüpfen. X.509-Zertifikate werden von vertrauenswürdigen Zertifizierungsstellen (Certificate Authority, CA) herausgegeben. Die CA nutzen ein oder mehrere spezielle Zertifikate, die als CA-Zertifikate bezeichnet werden, zum Herausgeben der X.509-Zertifikate. Nur die Zertifizierungsstelle hat Zugriff auf CA-Zertifikate.

AWS IoT Richtlinien definieren die Anzahl der Operationen, die für AWS IoT Geräte zulässig sind. Insbesondere erlauben und verweigern sie den Zugriff auf und Operationen auf AWS IoT Core der AWS IoT Greengrass Datenebene, wie das Veröffentlichen von MQTT-Nachrichten und das Abrufen von Geräteschatten.

Alle Geräte benötigen einen Eintrag in der AWS IoT Core Registrierung und ein aktiviertes X.509-Zertifikat mit einer angehängten Richtlinie. AWS IoT Geräte fallen in zwei Kategorien:

- **Greengrass-Kerne.** Greengrass-Core-Geräte verwenden Zertifikate und AWS IoT Richtlinien, um AWS IoT Core eine Verbindung herzustellen. Die Zertifikate und Richtlinien ermöglichen auch AWS IoT Greengrass die Bereitstellung von Konfigurationsinformationen, Lambda-Funktionen, Konnektoren und verwalteten Abonnements für Kerngeräte.
- **Client-Geräte.** Client-Geräte (auch verbundene Geräte, Greengrass-Geräte oder Geräte genannt) sind Geräte, die über MQTT eine Verbindung zu einem Greengrass-Core herstellen. Sie verwenden Zertifikate und Richtlinien, um sich mit dem Dienst zu AWS IoT Core verbinden. AWS IoT Greengrass Auf diese Weise können Client-Geräte den AWS IoT Greengrass Discovery Service verwenden, um ein Kerngerät zu finden und eine Verbindung zu diesem herzustellen. Ein Client-Gerät verwendet dasselbe Zertifikat, um eine Verbindung zum AWS IoT Core Geräte-Gateway und zum Kerngerät herzustellen. Client-Geräte verwenden außerdem Erkennungsinformationen für die gegenseitige Authentifizierung mit dem Kerngerät. Weitere Informationen erhalten Sie unter [the section called “Geräteverbindung – Workflow”](#) und [the section called “Verwalten der Geräteauthentifizierung mit dem Greengrass Core”](#).

X.509-Zertifikate

Die Kommunikation zwischen Core- und Client-Geräten sowie zwischen Geräten AWS IoT Core und/oder AWS IoT Greengrass muss authentifiziert werden. Diese gegenseitige Authentifizierung basiert auf registrierten X.509-Gerätezertifikaten und kryptografischen Schlüsseln.

In einer AWS IoT Greengrass Umgebung verwenden Geräte Zertifikate mit öffentlichen und privaten Schlüsseln für die folgenden Transport Layer Security (TLS) -Verbindungen:

- Die AWS IoT Client-Komponente auf dem Greengrass-Kern, die eine Verbindung zum AWS IoT Core und AWS IoT Greengrass über das Internet herstellt.
- Client-Geräte stellen eine Verbindung her AWS IoT Greengrass , um wichtige Erkennungsinformationen über das Internet abzurufen.

- Die MQTT-Serverkomponente auf dem Greengrass-Core, die über das lokale Netzwerk eine Verbindung zu Client-Geräten in der Gruppe herstellt.

Das AWS IoT Greengrass Core-Gerät speichert Zertifikate an zwei Orten:

- Kern-Gerätezertifikat in `/greengrass-root/certs`. In der Regel wird das Kern-Gerätezertifikat als `hash.cert.pem` bezeichnet (z. B. `86c84488a5.cert.pem`). Dieses Zertifikat wird vom AWS IoT Client für die gegenseitige Authentifizierung verwendet, wenn der Core eine Verbindung zu den AWS IoT Core AWS IoT Greengrass AND-Diensten herstellt.
- MQTT-Serverzertifikat in `/greengrass-root/ggc/var/state/server`. Das MQTT-Serverzertifikat hat den Namen `server.crt`. Dieses Zertifikat wird für die gegenseitige Authentifizierung zwischen dem lokalen MQTT-Server (auf dem Greengrass-Kern) und Greengrass-Geräten verwendet.

 Note

`greengrass-root` stellt den Pfad dar, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

Weitere Informationen finden Sie unter [the section called “Sicherheitsprinzipale”](#).

CA-Zertifikate

Core-Geräte und Client-Geräte laden ein Root-CA-Zertifikat herunter, das für die Authentifizierung mit AWS IoT Core und AWS IoT Greengrass -Diensten verwendet wird. Es wird empfohlen, ein Amazon Trust Services (ATS) CA-Stammzertifikat wie [Amazon Root CA 1](#) zu verwenden. Weitere Informationen finden Sie unter [CA-Zertifikate für die Serverauthentifizierung](#) im AWS IoT Core - Entwicklerhandbuch.

 Note

Ihr Root-CA-Zertifikatstyp muss mit Ihrem Endpunkt übereinstimmen. Verwenden Sie ein ATS-Root-CA-Zertifikat mit einem ATS-Endpunkt (bevorzugt) oder ein VeriSign Root-CA-Zertifikat mit einem älteren Endpunkt. Nur einige Amazon Web Services Services-Regionen unterstützen ältere Endgeräte. Weitere Informationen finden Sie unter [the section called “Dienstendpunkte müssen dem Zertifikatstyp entsprechen”](#).

Client-Geräte laden auch das CA-Zertifikat der Greengrass-Gruppe herunter. Dies wird verwendet, um das MQTT-Serverzertifikat auf dem Greengrass-Kern während der gegenseitigen Authentifizierung zu validieren. Weitere Informationen finden Sie unter [the section called “Geräteverbindung – Workflow”](#). Der Standardablaufzeitraum des MQTT-Serverzertifikats beträgt sieben Tage.

Zertifikatrotation auf dem lokalen MQTT-Server

Client-Geräte verwenden das lokale MQTT-Serverzertifikat für die gegenseitige Authentifizierung mit dem Greengrass-Core-Gerät. Standardmäßig läuft dieses Zertifikat nach sieben Tagen ab. Dieser begrenzte Zeitraum basiert auf bewährten Sicherheitsmethoden. Das MQTT-Serverzertifikat wird von dem CA-Gruppenzertifikat signiert, das in der Cloud gespeichert ist.

Damit die Zertifikatsrotation stattfinden kann, muss Ihr Greengrass-Core-Gerät online sein und regelmäßig direkt auf den AWS IoT Greengrass Service zugreifen können. Wenn das Zertifikat abläuft, versucht das Kerngerät, eine Verbindung zum AWS IoT Greengrass Dienst herzustellen, um ein neues Zertifikat zu erhalten. Wenn eine Verbindung hergestellt wurde, lädt das Kerngerät das neue MQTT-Serverzertifikat herunter und startet den lokalen MQTT-Service neu. Zu diesem Zeitpunkt sind alle Client-Geräte, die mit dem Core verbunden sind, getrennt. Wenn das Core-Gerät zum Zeitpunkt des Ablaufs offline ist, erhält es kein Ersatzzertifikat. Alle neuen Versuche, eine Verbindung mit dem Core-Gerät herzustellen, werden abgelehnt. Bestehende Verbindungen bleiben davon unberührt. Client-Geräte können keine Verbindung zum Kerngerät herstellen, bis die Verbindung zum AWS IoT Greengrass Dienst wiederhergestellt ist und ein neues MQTT-Serverzertifikat heruntergeladen werden kann.

Die Zeit bis zum Ablauf kann je nach Ihren Anforderungen auf einen Wert zwischen 7 und 30 Tagen festgelegt werden. Häufigere Rotationen erfordern häufigere Cloud-Verbindungen. Weniger häufige Rotationen können Sicherheitsrisiken darstellen. Wenn Sie das Ablaufdatum des Zertifikats auf einen Wert von mehr als 30 Tagen festlegen möchten, wenden Sie sich an Support.

In der AWS IoT Konsole können Sie das Zertifikat auf der Einstellungsseite der Gruppe verwalten. In der AWS IoT Greengrass API können Sie die [UpdateGroupCertificateConfiguration](#)Aktion verwenden.

Wenn das MQTT-Serverzertifikat abgelaufen ist, schlägt jeder Validierungsversuch für das Zertifikat fehl. Client-Geräte müssen in der Lage sein, den Fehler zu erkennen und die Verbindung zu beenden.

AWS IoT Richtlinien für den Betrieb auf Datenebene

Verwenden Sie AWS IoT Richtlinien, um den Zugriff auf die AWS IoT Core AWS IoT Greengrass Datenebene zu autorisieren. Die AWS IoT Core Datenebene besteht aus Vorgängen für Geräte, Benutzer und Anwendungen, z. B. das Herstellen einer Verbindung zu Themen AWS IoT Core und das Abonnieren von Themen. Die Datenebene AWS IoT Greengrass besteht aus Operationen für Greengrass-Geräte, z. B. dem Abrufen von Bereitstellungen und dem Aktualisieren von Konnektivitätsinformationen.

Eine AWS IoT Richtlinie ist ein JSON-Dokument, das einer [IAM-Richtlinie](#) ähnelt. Sie enthält eine oder mehrere Richtlinienanweisungen, die die folgenden Eigenschaften angeben:

- **Effect.** Der Zugriffsmodus, der Allow oder sein kann Deny.
- **Action.** Die Liste der Aktionen, die von der Richtlinie zugelassen oder verweigert werden.
- **Resource.** Die Liste der Ressourcen, für die die Aktion zulässig oder verweigert ist.

AWS IoT Richtlinien unterstützen * als Platzhalterzeichen und behandeln MQTT-Platzhalterzeichen (+und#) als wörtliche Zeichenketten. Weitere Informationen zum Platzhalter finden Sie unter [Verwenden von * Platzhaltern in Ressourcen im Benutzerhandbuch](#). ARNs AWS Identity and Access Management

Weitere Informationen finden Sie unter [AWS IoT -Richtlinien](#) und [AWS IoT -Richtlinienaktionen](#) im AWS IoT Core -Entwicklerhandbuch.

Note

AWS IoT Core ermöglicht es Ihnen, AWS IoT Richtlinien an Dinggruppen anzuhängen, um Berechtigungen für Gerätegruppen zu definieren. Dinggruppenrichtlinien ermöglichen keinen Zugriff auf AWS IoT Greengrass Datenebenenoperationen. Um einer Sache den Zugriff auf einen Vorgang auf der AWS IoT Greengrass Datenebene zu gewähren, fügen Sie die Berechtigung zu einer AWS IoT Richtlinie hinzu, die Sie dem Zertifikat der Sache hinzufügen.

AWS IoT Greengrass politische Aktionen

Greengrass-Kernaktionen

AWS IoT Greengrass definiert die folgenden Richtlinienaktionen, die Greengrass-Core-Geräte in AWS IoT Richtlinien verwenden können:

`greengrass:AssumeRoleForGroup`

Erlaubnis für ein Greengrass-Core-Gerät, Anmeldeinformationen mithilfe der Lambda-Funktion des Token Exchange Service (TES) -Systems abzurufen. Die Berechtigungen, die mit den abgerufenen Anmeldeinformationen verknüpft sind, basieren auf der Richtlinie, die der konfigurierten Gruppenrolle angefügt ist.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Kerngerät versucht, Anmeldeinformationen abzurufen (vorausgesetzt, die Anmeldeinformationen werden nicht lokal zwischengespeichert).

`greengrass:CreateCertificate`

Berechtigung für ein Greengrass-Kerngerät zum Erstellen eines eigenen Serverzertifikats.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Kerngerät ein Zertifikat erstellt. Greengrass-Kerngeräte versuchen, ein Serverzertifikat beim ersten Ausführen, wenn sich die Konnektivitätsinformationen des Kerns ändern, und in festgelegten Rotationszeiträumen zu erstellen.

`greengrass:GetConnectivityInfo`

Berechtigung für ein Greengrass-Kerngerät, seine eigenen Konnektivitätsinformationen abzurufen.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Core-Gerät versucht, seine Verbindungsinformationen von AWS IoT Core abzurufen.

`greengrass:GetDeployment`

Berechtigung für ein Greengrass-Kerngerät zum Abrufen von Bereitstellungen.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Kerngerät versucht, Bereitstellungen und Bereitstellungsstatus aus der Cloud abzurufen.

`greengrass:GetDeploymentArtifacts`

Erlaubnis für ein Greengrass-Core-Gerät, Bereitstellungsartefakte wie Gruppeninformationen oder Lambda-Funktionen abzurufen.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Kerngerät eine Bereitstellung erhält und dann versucht, Bereitstellungsartefakte abzurufen.

`greengrass:UpdateConnectivityInfo`

Berechtigung für ein Greengrass-Kerngerät, seine eigenen Konnektivitätsinformationen mit IP- oder Hostnameninformationen zu aktualisieren.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Kerngerät versucht, seine Konnektivitätsinformationen in der Cloud zu aktualisieren.

`greengrass:UpdateCoreDeploymentStatus`

Berechtigung für ein Greengrass-Kerngerät, den Status einer Bereitstellung zu aktualisieren.

Diese Berechtigung wird überprüft, wenn ein Greengrass-Kerngerät eine Bereitstellung erhält und dann versucht, den Bereitstellungsstatus zu aktualisieren.

Greengrass-Geräteaktionen

AWS IoT Greengrass definiert die folgende Richtlinienaktion, die Client-Geräte in AWS IoT Richtlinien verwenden können:

`greengrass:Discover`

Genehmigung für ein Client-Gerät, die [Discovery-API](#) zum Abrufen der wichtigsten Verbindungsinformationen und der Gruppensertifizierungsstelle seiner Gruppe zu verwenden.

Diese Berechtigung wird überprüft, wenn ein Client-Gerät die Discovery-API mit gegenseitiger TLS-Authentifizierung aufruft.

Minimale AWS IoT Richtlinie für das AWS IoT Greengrass Kerngerät

Die folgende Beispielrichtlinie enthält den Mindestsatz von Aktionen, die erforderlich sind, um eine grundlegende Greengrass-Funktionalität für Ihr Core-Gerät zu unterstützen.

- Die Richtlinie listet die MQTT-Themen und Themenfilter auf, für die das Core-Gerät Nachrichten veröffentlichen, abonnieren und empfangen kann, einschließlich Themen für den Schattenstatus. Um den Nachrichtenaustausch zwischen AWS IoT Core Lambda-Funktionen, Konnektoren und Client-Geräten in der Greengrass-Gruppe zu unterstützen, geben Sie die Themen und

Themenfilter an, die Sie zulassen möchten. Weitere Informationen finden Sie unter [Beispiele für Veröffentlichungs-/Abonnement-Richtlinien](#) im AWS IoT Core -Entwicklerhandbuch.

- Die Richtlinie umfasst einen Abschnitt, der es ermöglicht, AWS IoT Core den Schatten des Kerngeräts abzurufen, zu aktualisieren und zu löschen. Um die Shadow-Synchronisierung für Client-Geräte in der Greengrass-Gruppe zu ermöglichen, geben Sie die Amazon-Zielfressourcennamen (ARNs) in der Resource Liste an (z. B. `arn:aws:iot:region:account-id:thing/device-name`).
- Die Verwendung von [Ding-Richtlinienvariablen](#) (`iot:Connection.Thing.*`) in der AWS IoT Richtlinie für ein Kerngerät wird nicht unterstützt. Der Core verwendet dasselbe Gerätezertifikat, um [mehrere Verbindungen](#) herzustellen, AWS IoT Core aber die Client-ID in einer Verbindung entspricht möglicherweise nicht exakt dem Namen des Kern-Dings.
- Für die `greengrass:UpdateCoreDeploymentStatus`-Berechtigung ist das letzte Segment im Resource-ARN der URL-codierte ARN des Core-Geräts.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/core-name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-name-
*"
      ]
    }
  ],
}
```

```

    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot>DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:AssumeRoleForGroup",
        "greengrass:CreateCertificate"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:GetDeployment"
      ],
      "Resource": [
        "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/group-id/deployments/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [

```

```

        "greengrass:GetDeploymentArtifacts"
    ],
    "Resource": [
        "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/group-id/deployments/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "greengrass:UpdateCoreDeploymentStatus"
    ],
    "Resource": [
        "arn:aws:greengrass:us-east-1:123456789012:/
greengrass/groups/group-id/deployments/*/cores/arn%3Aaws%3Aiot
%3Aregion%3A123456789012%3Athing%2Fcore-name"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "greengrass:GetConnectivityInfo",
        "greengrass:UpdateConnectivityInfo"
    ],
    "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
    ]
}
]
}

```

Note

AWS IoT Richtlinien für Client-Geräte erfordern in der Regel ähnliche Berechtigungen für `iot:Connect`, `iot:Publish`, `iot:Receive`, und `iot:Subscribe` Aktionen. Damit ein Client-Gerät automatisch Verbindungsinformationen für die Kerne in den Greengrass-Gruppen erkennen kann, zu denen das Gerät gehört, muss die AWS IoT Richtlinie für ein Client-Gerät die `greengrass:Discover` Aktion beinhalten. Geben Sie in Resource diesem Abschnitt den ARN des Client-Geräts an, nicht den ARN des Greengrass-Core-Geräts. Beispiel:

```
{
  "Effect": "Allow",
  "Action": [
    "greengrass:Discover"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/device-name"
  ]
}
```

Die AWS IoT Richtlinie für Client-Geräte erfordert normalerweise keine Berechtigungen oder `iot:DeleteThingShadow` Aktionen `iot:GetThingShadow` `iot:UpdateThingShadow`, da der Greengrass-Kern Schattensynchronisierungsvorgänge für Client-Geräte abwickelt. Stellen Sie in diesem Fall sicher, dass der Resource Abschnitt für Shadow-Aktionen in der AWS IoT Core-Richtlinie auch die ARNs Client-Geräte umfasst.

In der AWS IoT Konsole können Sie die Richtlinie, die an das Zertifikat Ihres Cores angehängt ist, anzeigen und bearbeiten.

1. Erweitern Sie im Navigationsbereich unter Verwalten die Option Alle Geräte und wählen Sie dann Dinge aus.
2. Wählen Sie Ihren Kern aus.
3. Wählen Sie auf der Konfigurationsseite Ihres Cores den Tab Zertifikate aus.
4. Wählen Sie auf der Registerkarte Zertifikate Ihr Zertifikat aus.
5. Klicken Sie auf der Konfigurationsseite des Zertifikats auf Policies (Richtlinien) und wählen Sie dann die Richtlinie aus.

Wenn Sie die Richtlinie bearbeiten möchten, wählen Sie Aktive Version bearbeiten.

6. Überprüfen Sie die Richtlinie und fügen Sie nach Bedarf Berechtigungen hinzu, entfernen oder bearbeiten Sie sie.
7. Um eine neue Richtlinienversion als aktive Version festzulegen, wählen Sie unter Status der Richtlinienversion die Option Die bearbeitete Version als aktive Version für diese Richtlinie festlegen aus.
8. Wählen Sie Als neue Version speichern aus.

Identitäts- und Zugriffsmanagement für AWS IoT Greengrass

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service, den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IoT Greengrass IAM ist ein Programm AWS-Service, das Sie ohne zusätzliche Kosten nutzen können.

Note

In diesem Thema werden die Konzepte und Funktionen von IAM beschrieben. Informationen zu den IAM-Funktionen, die von unterstützt werden AWS IoT Greengrass, finden Sie unter [the section called “Wie AWS IoT Greengrass funktioniert mit IAM”](#)

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe [Behebung von Identitäts- und Zugriffsproblemen für AWS IoT Greengrass](#)).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe [Wie AWS IoT Greengrass funktioniert mit IAM](#)).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe [Beispiele für identitätsbasierte Richtlinien für AWS IoT Greengrass](#)).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden

finden Sie unter [So melden Sie sich bei Ihrem AWS-Konto an](#) im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter [AWS Signature Version 4 for API requests](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Tasks that require root user credentials](#) im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Erfordern, dass menschliche Benutzer für den Zugriff AWS mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter](#) verwenden müssen.

Eine [IAM-Gruppe](#) spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter [Anwendungsfälle für IAM-Benutzer](#) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie [von einer Benutzer- zu einer IAM-Rolle \(Konsole\) wechseln](#) AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter [Methoden, um eine Rolle zu übernehmen](#) im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter [Übersicht über JSON-Richtlinien](#) im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcen und unter welchen Bedingungen durchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien](#) im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter [Choose between managed policies and inline policies](#) im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie [einen Prinzipal angeben](#).

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF Weitere Informationen finden Sie unter [Übersicht über ACLs die Zugriffskontrollliste \(ACL\)](#) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- **Berechtigungsgrenzen** – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter [Berechtigungsgrenzen für IAM-Entitäten](#) im -IAM-Benutzerhandbuch.
- **Richtlinien zur Dienstkontrolle (SCPs)** — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter [Service-Kontrollrichtlinien](#) im AWS Organizations -Benutzerhandbuch.
- **Richtlinien zur Ressourcenkontrolle (RCPs)** — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter [Richtlinien zur Ressourcenkontrolle \(RCPs\)](#).
- **Sitzungsrichtlinien** – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter [Sitzungsrichtlinien](#) im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter [Bewertungslogik für Richtlinien](#) im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- [the section called “Wie AWS IoT Greengrass funktioniert mit IAM”](#)

- [the section called “Beispiele für identitätsbasierte Richtlinien”](#)
- [the section called “Problembehandlung bei Identitäts- und Zugriffsproblemen”](#)

Wie AWS IoT Greengrass funktioniert mit IAM

Bevor Sie IAM zur Verwaltung des Zugriffs auf verwenden AWS IoT Greengrass, sollten Sie sich mit den IAM-Funktionen vertraut machen, die Sie mit verwenden können. AWS IoT Greengrass

IAM-Feature	Von Greengrass unterstützt?
Identitätsbasierte Richtlinien mit Berechtigungen auf Ressourcenebene	Ja
Ressourcenbasierte Richtlinien	Nein
Zugriffskontrolllisten () ACLs	Nein
Auf Tags basierende Autorisierung	Nein
Temporäre Anmeldeinformationen	Ja
Service-verknüpfte Rollen	Nein
Servicerollen	Ja

Einen allgemeinen Überblick darüber, wie andere AWS Dienste mit IAM funktionieren, finden Sie im IAM-Benutzerhandbuch unter [AWS Dienste, die mit IAM funktionieren](#).

Identitätsbasierte Richtlinien für AWS IoT Greengrass

Mit identitätsbasierten IAM-Richtlinien können Sie zulässige oder verweigte Aktionen und Ressourcen sowie die Bedingungen angeben, unter denen Aktionen zugelassen oder verweigert werden. AWS IoT Greengrass unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Weitere Informationen zu allen Elementen, die Sie in einer Richtlinie verwenden, finden Sie unter [Referenz zu den IAM-JSON-Richtlinienelementen](#) im IAM-Benutzerhandbuch.

Aktionen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Bei Richtlinienaktionen wird das `greengrass:` Präfix vor der Aktion AWS IoT Greengrass verwendet. Um beispielsweise jemandem zu ermöglichen, die `ListGroups` API-Operation zu verwenden, um die Gruppen in seiner Liste aufzulisten AWS-Konto, nehmen Sie die `greengrass>ListGroups` Aktion in seine Richtlinie auf. Richtlinienanweisungen müssen ein `Action`- oder `NotAction`-Element enthalten. AWS IoT Greengrass definiert seinen eigenen Satz an Aktionen, die Aufgaben beschreiben, die Sie mit diesem Service durchführen können.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, listen Sie sie in Klammern (`[]`) auf und trennen Sie sie wie folgt durch Kommas:

```
"Action": [  
    "greengrass:action1",  
    "greengrass:action2",  
    "greengrass:action3"  
]
```

Sie können Platzhalter (`*`) verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `List` beginnen, einschließlich der folgenden Aktion:

```
"Action": "greengrass:List*"
```

Note

Es wird empfohlen, die Verwendung von Platzhaltern zu vermeiden, um alle verfügbaren Aktionen für einen Service anzugeben. Als bewährte Methode sollten Sie in einer Richtlinie die geringsten Berechtigungen mit eng begrenztem Umfang gewähren. Weitere Informationen finden Sie unter [the section called “Erteilen von Mindestberechtigungen”](#).

Die vollständige Liste der AWS IoT Greengrass [Aktionen finden Sie AWS IoT Greengrass im IAM-Benutzerhandbuch unter Definierte Aktionen von.](#)

Ressourcen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen [Amazon-Ressourcennamen \(ARN\)](#) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"

```

Die folgende Tabelle enthält die AWS IoT Greengrass Ressource ARNs, die im Resource Element einer Richtlinienerklärung verwendet werden kann. Eine Zuordnung der unterstützten Berechtigungen für AWS IoT Greengrass Aktionen auf Ressourcenebene finden Sie unter [Actions Defined by AWS IoT Greengrass](#) im IAM-Benutzerhandbuch.

Ressource	ARN
Group	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}</code>
GroupVersion	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}/versions/\${VersionId}</code>
CertificateAuthority	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}/certificateauthoriti es/\${CertificateAuthorityId}</code>
Deployment	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/groups/\${GroupId}/deployments/\${Deploy mentId}</code>
BulkDeployment	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/ greengrass/bulk/deployments/\${BulkDeploymentId}</code>

Ressource	ARN
<u>ConnectorDefinition</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/connectors/\${ConnectorDefinitionId}
<u>ConnectorDefinitionVersion</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/connectors/\${ConnectorDefinitionId}/versions/\${VersionId}
<u>CoreDefinition</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/cores/\${CoreDefinitionId}
<u>CoreDefinitionVersion</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/cores/\${CoreDefinitionId}/versions/\${VersionId}
<u>DeviceDefinition</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/devices/\${DeviceDefinitionId}
<u>DeviceDefinitionVersion</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/devices/\${DeviceDefinitionId}/versions/\${VersionId}
<u>FunctionDefinition</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/functions/\${FunctionDefinitionId}
<u>FunctionDefinitionVersion</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/functions/\${FunctionDefinitionId}/versions/\${VersionId}
<u>LoggerDefinition</u>	arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/loggers/\${LoggerDefinitionId}

Ressource	ARN
LoggerDefinitionVersion	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/loggers/\${LoggerDefinitionId}/versions/\${VersionId}</code>
ResourceDefinition	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/resources/\${ResourceDefinitionId}</code>
ResourceDefinitionVersion	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/resources/\${ResourceDefinitionId}/versions/\${VersionId}</code>
SubscriptionDefinition	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/subscriptions/\${SubscriptionDefinitionId}</code>
SubscriptionDefinitionVersion	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/definition/subscriptions/\${SubscriptionDefinitionId}/versions/\${VersionId}</code>
ConnectivityInfo	<code>arn:\${Partition}:greengrass:\${Region}:\${Account}:/greengrass/things/\${ThingName}/connectivityInfo</code>

Das folgende Resource Beispielement spezifiziert den ARN einer Gruppe in der Region USA West (Oregon) in AWS-Konto 123456789012:

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

Oder, um alle Gruppen anzugeben, die zu einer AWS-Konto bestimmten Gruppe gehören AWS-Region, verwenden Sie den Platzhalter anstelle der Gruppen-ID:

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

Einige AWS IoT Greengrass Aktionen (z. B. einige Listenoperationen) können für eine bestimmte Ressource nicht ausgeführt werden. In diesen Fällen müssen Sie allein den Platzhalter verwenden.


```
"Resource": ""
```

Um mehrere Ressourcen ARNs in einer Anweisung anzugeben, listen Sie sie in Klammern ([]) auf und trennen Sie sie durch Kommas, wie folgt:

```
"Resource": [  
    "resource-arn1",  
    "resource-arn2",  
    "resource-arn3"  
]
```

Weitere Informationen zu ARN-Formaten finden Sie unter [Amazon Resource Names \(ARNs\) und AWS Service-Namespaces](#) in der. Allgemeine Amazon Web Services-Referenz

Bedingungsschlüssel

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element Condition gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die [Bedingungsoperatoren](#) verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter [Kontextschlüssel für AWS globale Bedingungen](#) im IAM-Benutzerhandbuch.

AWS IoT Greengrass unterstützt die folgenden globalen Bedingungsschlüssel.

Key (Schlüssel)	Description
aws:CurrentTime	Filtert den Zugriff, indem die date/time Bedingungen für das aktuelle Datum und die aktuelle Uhrzeit überprüft werden.
aws:EpochTime	Filtert den Zugriff, indem die date/time Bedingungen für das aktuelle Datum und die aktuelle Uhrzeit in Epochen- oder Unix-Zeit überprüft werden.
aws:MultiFactorAuthAge	Filtert den Zugriff durch Prüfen, wie lange es her ist (in Sekunden), dass die MFA-validierten Sicherheitsanmelde

Key (Schlüssel)	Description
	daten, die die Anforderung getätigt haben, mittels Multi-Factor Authentication (MFA) ausgegeben wurden.
<code>aws:MultiFactorAuthPresent</code>	Filtert den Zugriff durch Überprüfen, ob Multi-Factor Authentication (MFA) zur Validierung der temporären Sicherheitsanmeldedaten der aktuellen Anforderung verwendet wurde.
<code>aws:RequestTag/\${TagKey}</code>	Filtert Anfragen zum Erstellen basierend auf den zulässigen Wertesätzen für jedes einzelne obligatorische Tag.
<code>aws:ResourceTag/\${TagKey}</code>	Filtert Aktionen basierend auf dem Tag-Wert, der der Ressource zugeordnet ist.
<code>aws:SecureTransport</code>	Filtert den Zugriff durch Überprüfen, ob die Anforderung mithilfe von SSL gesendet wurde.
<code>aws:TagKeys</code>	Filtert Anfragen zum Erstellen basierend auf dem Vorhandensein obligatorischer Tags in der Anfrage.
<code>aws:UserAgent</code>	Filtert den Zugriff nach der Client-Anwendung des Anforderers.

Weitere Informationen finden Sie unter [Globale AWS -Bedingungskontextschlüssel](#) im IAM-Benutzerhandbuch.

Beispiele

Beispiele für AWS IoT Greengrass identitätsbasierte Richtlinien finden Sie unter [the section called "Beispiele für identitätsbasierte Richtlinien"](#)

Ressourcenbasierte Richtlinien für AWS IoT Greengrass

AWS IoT Greengrass unterstützt keine [ressourcenbasierten](#) Richtlinien.

Zugriffskontrolllisten (ACLs)

AWS IoT Greengrass unterstützt nicht [ACLs](#).

Autorisierung auf der Grundlage von AWS IoT Greengrass Tags

AWS IoT Greengrass unterstützt keine Tag-basierte Autorisierung.

IAM-Rollen für AWS IoT Greengrass

Eine [IAM-Rolle](#) ist eine Entität innerhalb von Ihrem AWS-Konto, die über bestimmte Berechtigungen verfügt.

Verwenden temporärer Anmeldeinformationen mit AWS IoT Greengrass

Temporäre Anmeldeinformationen werden verwendet, um sich bei einem Verbund anzumelden, eine IAM-Rolle anzunehmen oder eine kontoübergreifende Rolle anzunehmen. Sie erhalten temporäre Sicherheitsanmeldedaten, indem Sie AWS STS API-Operationen wie [AssumeRole](#) oder [GetFederationToken](#) aufrufen.

Auf dem Greengrass-Kern werden temporäre Anmeldeinformationen für die [Gruppenrolle](#) für benutzerdefinierte Lambda-Funktionen und -Konnektoren verfügbar gemacht. Wenn Ihre Lambda-Funktionen das AWS SDK verwenden, müssen Sie keine Logik hinzufügen, um die Anmeldeinformationen abzurufen, da das AWS SDK dies für Sie erledigt.

Service-verknüpfte Rollen

AWS IoT Greengrass unterstützt keine [dienstbezogenen Rollen](#).

Servicerollen

Dieses Feature ermöglicht einem Service das Annehmen einer [Servicerolle](#) in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Diensten, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Dienstes beeinträchtigen.

AWS IoT Greengrass verwendet eine Servicerolle, um in Ihrem Namen auf einige Ihrer AWS Ressourcen zuzugreifen. Weitere Informationen finden Sie unter [the section called "Greengrass-Servicerolle"](#).

Auswahl einer IAM-Rolle in der Konsole AWS IoT Greengrass

In der AWS IoT Greengrass Konsole müssen Sie möglicherweise eine Greengrass-Servicerolle oder eine Greengrass-Gruppenrolle aus einer Liste von IAM-Rollen in Ihrem Konto auswählen.

- Die Greengrass-Servicerolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre AWS Ressourcen in anderen Diensten in Ihrem Namen. In der Regel müssen Sie die Servicerolle nicht auswählen, da die Konsole sie für Sie erstellen und konfigurieren kann. Weitere Informationen finden Sie unter [the section called “Greengrass-Servicerolle”](#).
- Die Greengrass-Gruppenrolle wird verwendet, um den Funktionen und Konnektoren von Greengrass Lambda in der Gruppe den Zugriff auf Ihre Ressourcen zu ermöglichen. AWS Sie kann auch AWS IoT Greengrass Berechtigungen zum Exportieren von Streams in AWS Dienste und zum Schreiben von Protokollen erteilen. CloudWatch Weitere Informationen finden Sie unter [the section called “Greengrass-Gruppenrolle.”](#).

Greengrass-Servicerolle

Die Greengrass-Servicerolle ist eine AWS Identity and Access Management (IAM) -Servicerolle, die den AWS IoT Greengrass Zugriff auf Ressourcen von AWS Diensten in Ihrem Namen autorisiert. Auf diese Weise können Sie wichtige Aufgaben ausführen AWS IoT Greengrass , z. B. das Abrufen Ihrer AWS Lambda Funktionen und das Verwalten von Schatten. AWS IoT

Um auf Ihre Ressourcen zugreifen AWS IoT Greengrass zu können, muss die Greengrass-Servicerolle mit Ihrer verknüpft AWS-Konto und AWS IoT Greengrass als vertrauenswürdige Entität angegeben werden. Die Rolle muss die [AWSGreengrassResourceAccessRolePolicy](#) verwaltete Richtlinie oder eine benutzerdefinierte Richtlinie enthalten, die entsprechende Berechtigungen für die von Ihnen verwendeten AWS IoT Greengrass Funktionen definiert. Diese Richtlinie wird von den Berechtigungen verwaltet AWS und definiert, die für den Zugriff auf Ihre AWS Ressourcen AWS IoT Greengrass verwendet werden.

Sie können dieselbe Greengrass-Servicerolle für mehrere AWS-Region s wiederverwenden, müssen sie jedoch an jedem AWS-Region Ort, an dem Sie sie verwenden AWS IoT Greengrass, Ihrem Konto zuordnen. Die Gruppenbereitstellung schlägt fehl, wenn die Servicerolle in der aktuellen Region AWS-Konto und Region nicht existiert.

In den folgenden Abschnitten wird beschrieben, wie Sie die Greengrass-Servicerolle im AWS-Managementkonsole oder AWS CLI erstellen und verwalten.

- [Verwaltung der Servicerolle \(Konsole\)](#)
- [Verwaltung der Servicerolle \(CLI\)](#)

 Note

Zusätzlich zu der Servicerolle, die den Zugriff auf Dienstebene autorisiert, können Sie einer Gruppe eine Gruppenrolle zuweisen. AWS IoT Greengrass Die Gruppenrolle ist eine separate IAM-Rolle, die steuert, wie die Funktionen und Konnektoren von Greengrass Lambda in der Gruppe auf Dienste zugreifen können. AWS

Verwalten der Greengrass-Servicerolle (Konsole)

Die AWS IoT Konsole erleichtert die Verwaltung Ihrer Greengrass-Servicerolle. Wenn Sie beispielsweise eine Greengrass-Gruppe erstellen oder bereitstellen, prüft die Konsole, ob Sie AWS-Konto einer Greengrass-Servicerolle in der aktuell in der AWS-Region Konsole ausgewählten Rolle zugeordnet sind. Andernfalls kann die Konsole eine Servicerolle für Sie erstellen und konfigurieren. Weitere Informationen finden Sie unter [the section called “Erstellen der Greengrass-Servicerolle”](#).

Sie können die AWS IoT Konsole für die folgenden Rollenverwaltungsaufgaben verwenden:

- [Suchen Ihrer Greengrass-Servicerolle](#)
- [Erstellen der Greengrass-Servicerolle](#)
- [Ändern der Greengrass-Servicerolle](#)
- [Trennen der Greengrass-Servicerolle](#)

 Note

Der Benutzer, der bei der Konsole angemeldet ist, muss über Berechtigungen zum Anzeigen, Erstellen oder Ändern der Servicerolle verfügen.

Suchen Ihrer Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um die Servicerolle zu finden, die aktuell verwendet AWS IoT Greengrass wird AWS-Region.

1. Wählen Sie im Navigationsbereich der [AWS IoT Konsole](#) Einstellungen aus.

2. Scrollen Sie zum Abschnitt Greengrass service role (Greengrass-Servicerolle), um Ihre Servicerolle und deren Richtlinien anzuzeigen.

Wenn Sie keine Servicerolle sehen, können Sie die Konsole eine für Sie erstellen oder konfigurieren lassen. Weitere Informationen finden Sie unter [Erstellen der Greengrass-Servicerolle](#).

Erstellen der Greengrass-Servicerolle (Konsole)

Die Konsole kann eine standardmäßige Greengrass-Servicerolle für Sie erstellen und konfigurieren. Diese Rolle hat die folgenden Eigenschaften.

Eigenschaft	Wert
Name	Greengrass_ServiceRole
Trusted entity (Vertrauenswürdige Entität)	AWS service: greengrass
Richtlinie	AWSGreengrassResourceAccessRolePolicy

Note

Wenn [Greengrass Device Setup](#) die Servicerolle erstellt, lautet der Rollename GreengrassServiceRole_*random-string*.

Wenn Sie eine Greengrass-Gruppe von der AWS IoT Konsole aus erstellen oder bereitstellen, prüft die Konsole, ob eine Greengrass-Servicerolle mit Ihrer AWS-Konto in der AWS-Region Konsole aktuell ausgewählten Person verknüpft ist. Falls nicht, werden Sie von der Konsole aufgefordert, in Ihrem Namen Lese- und Schreibzugriff auf AWS Dienste zuzulassen AWS IoT Greengrass .

Wenn Sie die Erlaubnis erteilen, überprüft die Konsole, ob eine Rolle mit dem Namen in Ihrer Greengrass_ServiceRole AWS-Konto vorhanden ist.

- Wenn die Rolle vorhanden ist, ordnet die Konsole die Servicerolle Ihrer AWS-Konto in der aktuellen AWS-Region Version zu.

- Wenn die Rolle nicht existiert, erstellt die Konsole eine standardmäßige Greengrass-Servicerolle und fügt sie Ihrer aktuellen Rolle AWS-Konto hinzu. AWS-Region

Note

Wenn Sie eine Servicerolle mit benutzerdefinierten Rollenrichtlinien erstellen möchten, verwenden Sie die IAM-Konsole, um die Rolle zu erstellen oder zu ändern. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen für einen AWS Dienst](#) oder [Ändern einer Rolle](#) im IAM-Benutzerhandbuch. Stellen Sie sicher, dass die Rolle Berechtigungen erteilt, die der verwalteten Richtlinie `AWSGreengrassResourceAccessRolePolicy` für die verwendeten Funktionen und Ressourcen entsprechen. Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel beschränken den Zugriff, sodass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie [Serviceübergreifende Confused-Deputy-Prävention](#).

Wenn Sie eine Servicerolle erstellen, kehren Sie zur AWS IoT Konsole zurück und ordnen Sie die Rolle der Gruppe zu. Sie können dies unter Greengrass service role (Greengrass-Servicerolle) auf der Seite Settings (Einstellungen) der Gruppe vornehmen.

Ändern der Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um eine andere Greengrass-Servicerolle auszuwählen, die Sie Ihrer AWS-Region aktuell AWS-Konto in der Konsole ausgewählten zuordnen möchten.

1. Wählen Sie im Navigationsbereich der [AWS IoT Konsole](#) Einstellungen aus.
2. Wählen Sie unter Greengrass-Servicerolle die Option Rolle ändern aus.

Das Dialogfeld „Greengrass-Serverolle aktualisieren“ wird geöffnet und zeigt die IAM-Rollen in Ihrem System an AWS-Konto, die AWS IoT Greengrass als vertrauenswürdige Entität definiert sind.

3. Wählen Sie die Greengrass-Servicerolle aus, die Sie zuordnen möchten.

4. Wählen Sie Rolle anhängen.

Note

Um der Konsole das Erstellen einer standardmäßigen Greengrass-Servicerolle für Sie zu erlauben, wählen Sie **Create role for me** (Rolle für mich erstellen) aus, anstatt eine Rolle aus der Liste auszuwählen. Der Link **Rolle für mich erstellen** wird nicht angezeigt, wenn sich eine benannte Rolle in Ihrer `Greengrass_ServiceRole` befindet AWS-Konto.

Trennen der Greengrass-Servicerolle (Konsole)

Gehen Sie wie folgt vor, um die Greengrass-Servicerolle von Ihrer AWS-Region aktuell AWS-Konto in der Konsole ausgewählten Rolle zu trennen. Dadurch werden die Berechtigungen für den Zugriff AWS IoT Greengrass auf AWS Dienste in der aktuellen Version aufgehoben. AWS-Region

Important

Durch das Trennen der Servicerolle können aktive Operationen unterbrochen werden.

1. Wählen Sie im Navigationsbereich der [AWS IoT Konsole](#) **Einstellungen** aus.
2. Wählen Sie unter Greengrass-Servicerolle die Option **Rolle trennen** aus.
3. Wählen Sie im Bestätigungsdialogfeld **Trennen** aus.

Note

Wenn Sie die Rolle nicht mehr benötigen, können Sie sie in der IAM-Konsole löschen. Weitere Informationen zum Löschen von Rollen finden Sie unter [Löschen von Rollen oder Instance-Profilen](#) im IAM-Benutzerhandbuch.

Andere Rollen ermöglichen möglicherweise AWS IoT Greengrass den Zugriff auf Ihre Ressourcen. Um alle Rollen zu finden, mit denen AWS IoT Greengrass Sie in Ihrem Namen Berechtigungen übernehmen können, suchen Sie in der IAM-Konsole auf der Seite **Rollen** in der Spalte **Vertrauenswürdige Entitäten** nach Rollen, die `AWS service: greengrass` enthalten.

Verwalten der Greengrass-Servicerolle (CLI)

Bei den folgenden Verfahren gehen wir davon aus, dass der für die Verwendung Ihrer AWS-Konto ID installiert und konfiguriert AWS CLI ist. Weitere Informationen finden Sie unter [Installation der AWS Befehlszeilenschnittstelle](#) und [Konfiguration von AWS CLI im AWS Command Line Interface Benutzerhandbuch](#).

Sie können das AWS CLI für die folgenden Rollenverwaltungsaufgaben verwenden:

- [Abrufen Ihrer Greengrass-Servicerolle](#)
- [Erstellen der Greengrass-Servicerolle](#)
- [Entfernen der Greengrass-Servicerolle](#)

Abrufen der Greengrass-Servicerolle (CLI)

Verwenden Sie das folgende Verfahren, um herauszufinden, ob eine Greengrass-Servicerolle mit Ihrer AWS-Konto in einem AWS-Region verknüpft ist.

- Rufen Sie die Servicerolle ab. *region* Ersetzen Sie durch Ihre AWS-Region (zum Beispiel *us-west-2*).

```
aws Greengrass get-service-role-for-account --region region
```

Wenn Ihrem Konto bereits eine Greengrass-Servicerolle zugewiesen ist, werden die folgenden Rollenmetadaten zurückgegeben.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Wenn keine Rollenmetadaten zurückgegeben werden, müssen Sie die Servicerolle erstellen (falls sie nicht existiert) und sie Ihrem Konto in der zuzuordnen AWS-Region.

Erstellen der Greengrass-Servicerolle (CLI)

Gehen Sie wie folgt vor, um eine Rolle zu erstellen und sie Ihrer Rolle zuzuordnen AWS-Konto.

Um die Servicerolle mit IAM zu erstellen

1. Erstellen Sie die Rolle mit einer Vertrauensrichtlinie, die es ermöglicht, die Rolle AWS IoT Greengrass zu übernehmen. In diesem Beispiel wird eine Rolle namens `Greengrass_ServiceRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel beschränken den Zugriff, sodass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie [Serviceübergreifende Confused-Deputy-Prävention](#).

Linux, macOS, 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": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}
```

Windows command prompt

```
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. Kopieren Sie den Rollen-ARN aus den Rollenmetadaten in der Ausgabe. Sie verknüpfen die Servicerolle mithilfe des ARN mit Ihrem Konto.
3. Fügen Sie der Rolle die AWSGreengrassResourceAccessRolePolicy-Richtlinie an.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

Um die Servicerolle mit Ihrem zu verknüpfen AWS-Konto

- Weisen Sie die Rolle Ihrem Konto zu. *role-arn* Ersetzen Sie durch die Dienstrolle *region* ARN und durch Ihre AWS-Region (z. B. us-west-2).

```
aws greengrass associate-service-role-to-account --role-arn role-arn --region region
```

Bei erfolgreicher Durchführung wird die folgende Meldung zurückgegeben.

```
{
  "AssociatedAt": "timestamp"
}
```

Entfernen der Greengrass-Servicerolle (CLI)

Gehen Sie wie folgt vor, um die Greengrass-Servicerolle von Ihrer zu trennen. AWS-Konto

- Haben Sie die Zuordnung der Servicerolle bei Ihrem Konto auf. *region* Ersetzen Sie durch Ihre AWS-Region (zum Beispiel). us-west-2

```
aws greengrass disassociate-service-role-from-account --region region
```

Bei erfolgreicher Durchführung wird die folgende Meldung zurückgegeben.

```
{
  "DisassociatedAt": "timestamp"
}
```

Note

Sie sollten die Servicerolle löschen, wenn Sie sie in keiner verwenden AWS-Region. Verwenden Sie zuerst [delete-role-policy](#), um die verwaltete AWSGreengrassResourceAccessRolePolicy-Richtlinie von der Rolle zu lösen, und verwenden Sie dann [delete-role](#), um die Rolle zu löschen. Weitere Informationen zum Löschen von Rollen finden Sie unter [Löschen von Rollen oder Instance-Profilen](#) im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- [Erstellen einer Rolle zum Delegieren von Berechtigungen für einen AWS Dienst](#) im IAM-Benutzerhandbuch
- [Ändern einer Rolle](#) im IAM-Benutzerhandbuch
- [Löschen von Rollen oder Instanzprofilen](#) im IAM-Benutzerhandbuch
- AWS IoT Greengrass Befehle in der AWS CLI Befehlsreferenz
 - [associate-service-role-to-Konto](#)
 - [disassociate-service-role-from-Konto](#)
 - [get-service-role-for-Konto](#)
- IAM-Befehle in der Befehlsreferenz AWS CLI
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

Greengrass-Gruppenrolle.

Die Greengrass-Gruppenrolle ist eine IAM-Rolle, die Code, der auf einem Greengrass-Kern ausgeführt wird, autorisiert, auf Ihre Ressourcen zuzugreifen. AWS Sie erstellen die Rolle und verwalten die Berechtigungen in AWS Identity and Access Management (IAM) und ordnen die Rolle Ihrer Greengrass-Gruppe zu. Eine Greengrass-Gruppe verfügt über eine Gruppenrolle. Um Berechtigungen hinzuzufügen oder zu ändern, können Sie eine andere Rolle hinzufügen oder die IAM-Richtlinien ändern, die der Rolle zugewiesen sind.

Die Rolle muss AWS IoT Greengrass als vertrauenswürdige Entität definiert werden. Abhängig von Ihrem Geschäftsszenario kann die Gruppenrolle IAM-Richtlinien enthalten, die Folgendes definieren:

- Berechtigungen für benutzerdefinierte [Lambda-Funktionen](#) für den Zugriff AWS auf Dienste.
- Berechtigungen für [Konnektoren](#) für den Zugriff AWS auf Dienste.
- Berechtigungen für [Stream Manager](#) zum Exportieren von Streams in AWS IoT Analytics und Kinesis Data Streams.
- Berechtigungen zum Zulassen der [CloudWatch -Protokollierung](#).

In den folgenden Abschnitten wird beschrieben, wie Sie eine Greengrass-Gruppenrolle im oder anhängen oder trennen. AWS-Managementkonsole AWS CLI

- [Verwalten der Gruppenrolle \(Konsole\)](#)
- [Verwalten der Gruppenrolle \(CLI\)](#)

Note

Zusätzlich zu der Gruppenrolle, die den Zugriff vom Greengrass-Kern aus autorisiert, können Sie eine [Greengrass-Servicerolle](#) zuweisen, die den AWS IoT Greengrass Zugriff auf AWS Ressourcen in Ihrem Namen ermöglicht.

Verwalten der Greengrass-Gruppenrolle (Konsole)

Sie können die AWS IoT Konsole für die folgenden Rollenverwaltungsaufgaben verwenden:

- [Suchen Ihrer Greengrass-Gruppenrolle](#)
- [Hinzufügen oder Ändern der Greengrass-Gruppenrolle](#)

- [Entfernen der Greengrass-Gruppenrolle](#)

 Note

Der Benutzer, der bei der Konsole angemeldet ist, muss über Berechtigungen zum Verwalten der Rolle verfügen.

Suchen Ihrer Greengrass-Gruppenrolle (Konsole)

Gehen Sie wie folgt vor, um die Rolle zu finden, die einer Greengrass-Gruppe zugeordnet ist.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option Einstellungen anzeigen aus.

Wenn der Gruppe eine Rolle zugewiesen ist, wird sie unter Gruppenrolle angezeigt.

Hinzufügen oder Ändern der Greengrass-Gruppenrolle (Konsole)

Gehen Sie wie folgt vor, um eine IAM-Rolle aus Ihrer auszuwählen AWS-Konto , die Sie einer Greengrass-Gruppe hinzufügen möchten.

Für eine Gruppenrolle gelten folgende Anforderungen:

- AWS IoT Greengrass als vertrauenswürdige Entität definiert.
- Die mit der Rolle verknüpften Berechtigungsrichtlinien müssen Ihren AWS Ressourcen die Berechtigungen gewähren, die für die Lambda-Funktionen und -Konnektoren in der Gruppe sowie für die Greengrass-Systemkomponenten erforderlich sind.

 Note

Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie

aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie [Serviceübergreifende Confused-Deputy-Prävention](#).

Verwenden Sie die IAM-Konsole, um die Rolle und ihre Berechtigungen zu erstellen und zu konfigurieren. Schritte zur Erstellung einer Beispielrolle, die den Zugriff auf eine Amazon DynamoDB-Tabelle ermöglicht, finden Sie unter [the section called “Konfigurieren der Gruppenrolle”](#). Allgemeine Schritte finden Sie unter [Erstellen einer Rolle für einen AWS Service \(Konsole\)](#) im IAM-Benutzerhandbuch.

Nachdem die Rolle konfiguriert wurde, verwenden Sie die AWS IoT Konsole, um die Rolle der Gruppe hinzuzufügen.

 Note

Dieses Verfahren ist nur erforderlich, um eine Rolle für die Gruppe auszuwählen. Es ist nicht erforderlich, nachdem Sie die Berechtigungen der aktuell ausgewählten Gruppenrolle geändert haben.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option Einstellungen anzeigen aus.
4. Wählen Sie unter Gruppenrolle aus, ob Sie die Rolle hinzufügen oder ändern möchten:
 - Um die Rolle hinzuzufügen, wählen Sie Associate role und wählen Sie dann Ihre Rolle aus Ihrer Rollenliste aus. Dies sind die Rollen in Ihrem System AWS-Konto, die AWS IoT Greengrass als vertrauenswürdige Entität definiert sind.
 - Um eine andere Rolle auszuwählen, wählen Sie Rolle bearbeiten und wählen Sie dann Ihre Rolle aus Ihrer Rollenliste aus.
5. Wählen Sie Speichern.

Entfernen der Greengrass-Gruppenrolle (Konsole)

Führen Sie die folgenden Schritte aus, um die Rolle von einer Greengrass-Gruppe zu trennen.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite die Option Einstellungen anzeigen aus.
4. Wählen Sie unter Gruppenrolle die Option Rolle trennen aus.
5. Wählen Sie im Bestätigungsdialogfeld die Option Rolle trennen aus. In diesem Schritt wird die Rolle aus der Gruppe entfernt, die Rolle wird jedoch nicht gelöscht. Wenn Sie die Rolle löschen möchten, verwenden Sie die IAM-Konsole.

Verwalten der Greengrass-Gruppenrolle (CLI)

Sie können die AWS CLI für die folgenden Rollenverwaltungsaufgaben verwenden:

- [Abrufen Ihrer Greengrass-Gruppenrolle](#)
- [Erstellen der Greengrass-Gruppenrolle](#)
- [Entfernen der Greengrass-Gruppenrolle](#)

Abrufen der Greengrass-Gruppenrolle (CLI)

Befolgen Sie diese Schritte, um herauszufinden, ob einer Greengrass-Gruppe eine Rolle zugewiesen ist.

1. Sie finden die ID der Zielgruppe in der Liste Ihrer Gruppen.

```
aws greengrass list-groups
```

Nachfolgend finden Sie eine `list-groups`-Beispielantwort. Jede Gruppe in der Antwort enthält eine `Id`-Eigenschaft, die die Gruppen-ID enthält.

```
{
```



```

"Groups": [
  {
    "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
    "Name": "MyFirstGroup",
    "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
    "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
    "CreationTimestamp": "2019-11-11T05:47:31.435Z",
    "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
    "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
  },
  {
    "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
    "Name": "GreenhouseSensors",
    "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
    "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
    "CreationTimestamp": "2020-01-07T19:58:36.774Z",
    "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
    "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
  },
  ...
]
}

```

Weitere Informationen, darunter Beispiele, die die `query`-Option zum Filtern von Ergebnissen verwenden, finden Sie unter [the section called “Abrufen der Gruppen-ID”](#).

2. Kopieren Sie die Id der Zielgruppe aus der Ausgabe.
3. Rufen Sie die Gruppenrolle ab. `group-id` Ersetzen Sie es durch die ID der Zielgruppe.

```
aws greengrass get-associated-role --group-id group-id
```

Wenn Ihrer Greengrass-Gruppe eine Rolle zugewiesen ist, werden die folgenden Rollenmetadaten zurückgegeben.

```

{
  "AssociatedAt": "timestamp",

```

```
"RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Wenn Ihrer Gruppe keine Rolle zugewiesen ist, wird der folgende Fehler zurückgegeben.

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
attach an IAM role to this deployment group.
```

Erstellen der Greengrass-Gruppenrolle (CLI)

Führen Sie die folgenden Schritte aus, um eine Rolle zu erstellen und sie einer Greengrass-Gruppe zuzuweisen.

Um die Gruppenrolle mit IAM zu erstellen

1. Erstellen Sie die Rolle mit einer Vertrauensrichtlinie, die es ermöglicht, die Rolle AWS IoT Greengrass zu übernehmen. In diesem Beispiel wird eine Rolle namens `MyGreengrassGroupRole` erstellt, aber Sie können einen anderen Namen verwenden. Wir empfehlen, dass Sie auch die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel in Ihre Vertrauensrichtlinie aufnehmen, um das Sicherheitsproblem „Confused Deputy“ zu vermeiden. Die Bedingungskontextschlüssel schränken den Zugriff so ein, dass nur Anfragen zugelassen werden, die vom angegebenen Konto und Greengrass-Workspace stammen. Weitere Informationen zum Confused-Deputy-Problem finden Sie [Serviceübergreifende Confused-Deputy-Prävention](#).

Linux, macOS, or Unix

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
```

```

    "StringEquals": {
      "aws:SourceAccount": "account-id"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/
groups/group-id"
    }
  }
}
]
}'

```

Windows command prompt

```

aws iam create-role --role-name MyGreengrassGroupRole --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:/greengrass/groups/group-id\"},
\"StringEquals\": {\"aws:SourceAccount\": \"account-id\"}}]}]"

```

2. Kopieren Sie den Rollen-ARN aus den Rollenmetadaten in der Ausgabe. Sie verknüpfen die Rolle mithilfe des ARN mit Ihrer Gruppe .
3. Fügen Sie der Rolle verwaltete oder Inline-Richtlinien zur Unterstützung Ihrer geschäftlichen Situation an. Wenn beispielsweise eine benutzerdefinierte Lambda-Funktion aus Amazon S3 liest, können Sie die AmazonS3ReadOnlyAccess verwaltete Richtlinie an die Rolle anhängen.

```

aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess

```

Wenn dies erfolgreich ist, wird keine Antwort zurückgegeben.

So verknüpfen Sie die Rolle mit Ihrer Greengrass-Gruppe:

1. Sie finden die ID der Zielgruppe in der Liste Ihrer Gruppen.

```

aws greengrass list-groups

```

Nachfolgend finden Sie eine `list-groups`-Beispielantwort. Jede Gruppe in der Antwort enthält eine `Id`-Eigenschaft, die die Gruppen-ID enthält.

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "CreationTimestamp": "2020-01-07T19:58:36.774Z",
      "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
  ]
}
```

Weitere Informationen, darunter Beispiele, die die `query`-Option zum Filtern von Ergebnissen verwenden, finden Sie unter [the section called “Abrufen der Gruppen-ID”](#).

2. Kopieren Sie die `Id` der Zielgruppe aus der Ausgabe.
3. Weisen Sie die Rolle zu Ihrer Gruppe zu. `group-id` Ersetzen Sie durch die ID der Zielgruppe und `role-arn` durch den ARN der Gruppenrolle.

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

Bei erfolgreicher Durchführung wird die folgende Meldung zurückgegeben.

```
{
  "AssociatedAt": "timestamp"
}
```

Entfernen der Greengrass-Gruppenrolle (CLI)

Führen Sie die folgenden Schritte aus, um die Gruppenrolle von Ihrer Greengrass-Gruppe zu trennen.

1. Sie finden die ID der Zielgruppe in der Liste Ihrer Gruppen.

```
aws greengrass list-groups
```

Nachfolgend finden Sie eine `list-groups`-Beispielantwort. Jede Gruppe in der Antwort enthält eine `Id`-Eigenschaft, die die Gruppen-ID enthält.

```
{
  "Groups": [
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "Name": "MyFirstGroup",
      "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
      "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
      "CreationTimestamp": "2019-11-11T05:47:31.435Z",
      "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
      "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE"
    },
    {
      "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
      "Name": "GreenhouseSensors",
      "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
      "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
    }
  ]
}
```

```
        "CreationTimestamp": "2020-01-07T19:58:36.774Z",
        "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
        "Arn": "arn:aws:us-west-2:123456789012:greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
    },
    ...
]
}
```

Weitere Informationen, darunter Beispiele, die die `query`-Option zum Filtern von Ergebnissen verwenden, finden Sie unter [the section called “Abrufen der Gruppen-ID”](#).

2. Kopieren Sie die Id der Zielgruppe aus der Ausgabe.
3. Trennen Sie die Rolle von Ihrer Gruppe. `group-id` Ersetzen Sie durch die ID der Zielgruppe.

```
aws greengrass disassociate-role-from-group --group-id group-id
```

Bei erfolgreicher Durchführung wird die folgende Meldung zurückgegeben.

```
{
  "DisassociatedAt": "timestamp"
}
```

Note

Sie können die Gruppenrolle löschen, wenn Sie sie nicht verwenden. Verwenden Sie zuerst [delete-role-policy](#), um alle verwalteten Richtlinien von der Rolle zu trennen, und dann [delete-role](#), um die Rolle zu löschen. Weitere Informationen zum Löschen von Rollen finden Sie unter [Löschen von Rollen oder Instance-Profilen](#) im IAM-Benutzerhandbuch.

Weitere Informationen finden Sie auch unter

- Verwandte Themen im IAM-Benutzerhandbuch
 - [Eine Rolle erstellen, um Berechtigungen an einen Dienst zu delegieren AWS](#)
 - [Ändern einer Rolle](#)
 - [Hinzufügen und Entfernen von IAM-Identitätsberechtigungen](#)

- [Löschen von Rollen oder Instance-Profilen](#)
- AWS IoT Greengrass Befehle in der AWS CLI Befehlsreferenz
 - [list-groups](#)
 - [associate-role-to-group](#)
 - [disassociate-role-from-group](#)
 - [get-associated-role](#)
- IAM-Befehle in der AWS CLI Befehlsreferenz
 - [attach-role-policy](#)
 - [create-role](#)
 - [delete-role](#)
 - [delete-role-policy](#)

Serviceübergreifende Confused-Deputy-Prävention

Das Problem des verwirrten Stellvertreters ist ein Sicherheitsproblem, bei dem eine Entität, die keine Berechtigung zur Durchführung einer Aktion hat, eine privilegiere Entität zur Durchführung der Aktion zwingen kann. In AWS kann ein dienstübergreifendes Identitätswechsels zu einem Problem mit dem verwirrten Stellvertreter führen. Ein dienstübergreifender Identitätswechsel kann auftreten, wenn ein Dienst (der Anruf-Dienst) einen anderen Dienst anruft (den aufgerufenen Dienst). Der aufrufende Service kann manipuliert werden, um seine Berechtigungen zu verwenden, um Aktionen auf die Ressourcen eines anderen Kunden auszuführen, für die er sonst keine Zugriffsberechtigung haben sollte. Um dies zu verhindern, bietet AWS Tools, mit denen Sie Ihre Daten für alle Services mit Serviceprinzipalen schützen können, die Zugriff auf Ressourcen in Ihrem Konto erhalten haben.

Wir empfehlen, die Kontextschlüssel [aws:SourceArn](#) und die [aws:SourceAccount](#) globalen Bedingungsschlüssel in Ressourcenrichtlinien zu verwenden, um die Berechtigungen einzuschränken, die der AWS IoT Greengrass Ressource einen anderen Dienst gewähren. Wenn Sie beide globalen Bedingungskontextschlüssel verwenden, müssen der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert dieselbe Konto-ID verwenden, wenn sie in derselben Richtlinienanweisung verwendet werden.

Der Wert von `aws:SourceArn` muss die Greengrass-Kundenressource sein, die der `sts:AssumeRole` Anfrage zugeordnet ist.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontext-Schlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. Wenn Sie den vollständigen ARN der Ressource nicht kennen oder wenn Sie mehrere Ressourcen angeben, verwenden Sie den globalen Bedingungskontext-Schlüssel `aws:SourceArn` mit Platzhaltern (*) für die unbekannten Teile des ARN. Beispiel, `arn:aws:greengrass:region:account-id:*`.

Beispiele für Richtlinien, die die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globalen Bedingungsschlüssel verwenden, finden Sie in den folgenden Themen:

- [Erstellen der Greengrass-Servicerolle](#)
- [Erstellen der Greengrass-Gruppenrolle](#)
- [Erstellen und konfigurieren Sie eine IAM-Ausführungsrolle für Massenbereitstellungen](#)

Beispiele für identitätsbasierte Richtlinien für AWS IoT Greengrass

IAM-Benutzer besitzen keine Berechtigungen zum Erstellen oder Ändern von AWS IoT Greengrass -Ressourcen. Sie können auch keine Aufgaben mithilfe der API AWS-Managementkonsole AWS CLI, oder AWS ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den IAM-Benutzern oder -Gruppen anfügen, die diese Berechtigungen benötigen.

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand AWS IoT Greengrass Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter [Von AWS verwaltete Richtlinien](#) oder [Von AWS verwaltete Richtlinien für Auftragsfunktionen](#) im IAM-Benutzerhandbuch.

- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter [Richtlinien und Berechtigungen in IAM](#) im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter [IAM-JSON-Richtlinienelemente: Bedingung](#) im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter [Richtlinienvvalidierung mit IAM Access Analyzer](#) im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter [Sicherer API-Zugriff mit MFA](#) im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter [Best Practices für die Sicherheit in IAM](#) im IAM-Benutzerhandbuch.

AWS verwaltete Richtlinien für AWS IoT Greengrass

AWS IoT Greengrass verwaltet die folgenden AWS verwalteten Richtlinien, mit denen Sie IAM-Benutzern und -Rollen Berechtigungen gewähren können.

Richtlinie	Description
<u>AWSGreengrassFullAccess</u>	Ermöglicht alle AWS IoT Greengrass Aktionen für all Ihre AWS Ressourcen. Diese Richtlinie wird für AWS IoT Greengrass <u>Dienstadministratoren</u> oder zu Testzwecken empfohlen.
<u>AWSGreengrassReadOnlyAccess</u>	Get AWS IoT Greengrass Genehmigungen List und Aktionen für all Ihre AWS Ressourcen.
<u>AWSGreengrassResourceAccessRolePolicy</u>	Ermöglicht den Zugriff auf Ressourcen von AWS Diensten wie AWS Lambda AWS IoT Device Shadow. Dies ist die Standardrichtlinie, die für die <u>Greengrass-Servicerolle</u> verwendet wird. Diese Richtlinie dient allgemein zur Erleichterung des Zugriffs. Sie können eine benutzerdefinierte Richtlinie einrichten, die restriktiver ist.
<u>Greengrass OTAUpdate ArtifactAccess</u>	Ermöglicht den schreibgeschützten Zugriff auf over-the-air (OTA-) Aktualisierungsartefakte für die AWS IoT Greengrass Core-Software in allen s. AWS-Region

Beispiele für Richtlinien

Im folgenden Beispiel erteilen benutzerdefinierte Richtlinien Berechtigungen für häufig auftretende Szenarien.

Beispiele

- [Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer](#)

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter [Erstellen von Richtlinien auf der JSON-Registerkarte](#) im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der OR-API. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Behebung von Identitäts- und Zugriffsproblemen für AWS IoT Greengrass

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS IoT Greengrass und IAM auftreten können.

Problembereiche

- [Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT Greengrass](#)
- [Fehler: Greengrass ist nicht zur Übernahme der Servicerolle berechtigt, die mit diesem Konto verknüpft ist; oder der Fehler: Fehlgeschlagen: TES-Servicerolle ist nicht mit diesem Konto verknüpft.](#)
- [<account-id><role-name><region>Fehler: Beim Versuch, die Rolle arn:aws:iam: :role/ für den Zugriff auf die S3-URL https://-greengrass-updates.s3 zu verwenden, wurde die Berechtigung verweigert. <region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.](#)
- [Der Geräteschatten wird nicht mit der Cloud synchronisiert.](#)
- [Ich bin nicht berechtigt, IAM auszuführen: PassRole](#)
- [Ich bin Administrator und möchte anderen Zugriff gewähren AWS IoT Greengrass](#)
- [Ich möchte Personen außerhalb von mir den Zugriff auf meine Ressourcen ermöglichen AWS-Konto AWS IoT Greengrass](#)

Hilfe zur allgemeinen Problembehandlung finden Sie unter [Fehlerbehebung](#).

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT Greengrass

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung einer Aktion autorisiert sind, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort bereitgestellt hat.

Der folgende Beispielfehler tritt auf, wenn der mateojackson IAM-Benutzer versucht, Details zu einer Core-Definitionsversion anzuzeigen, aber nicht über die greengrass:GetCoreDefinitionVersion entsprechenden Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

In diesem Fall bittet Mateo seinen Administrator um die Aktualisierung seiner Richtlinien, um unter Verwendung der Aktion `arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` auf die Ressource `greengrass:GetCoreDefinitionVersion` zugreifen zu können.

Fehler: Greengrass ist nicht zur Übernahme der Servicerolle berechtigt, die mit diesem Konto verknüpft ist; oder der Fehler: Fehlgeschlagen: TES-Servicerolle ist nicht mit diesem Konto verknüpft.

Lösung: Möglicherweise wird Ihnen dieser Fehler angezeigt, wenn die Bereitstellung fehlschlägt. Vergewissern Sie sich, dass Ihrer aktuellen AWS-Region Greengrass-Servicerolle eine AWS-Konto Greengrass-Servicerolle zugeordnet ist. Für weitere Informationen siehe [the section called “Verwaltung der Servicerolle \(CLI\)”](#) oder [the section called “Verwaltung der Servicerolle \(Konsole\)”](#).

`<account-id><role-name><region>` Fehler: Beim Versuch, die Rolle `arn:aws:iam: :role/` für den Zugriff auf die S3-URL `https://-greengrass-updates.s3` zu verwenden, wurde die Berechtigung verweigert. `<region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz`.

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn ein (OTA-) Update fehlschlägt. over-the-air Fügen Sie in der Rollenrichtlinie für den Unterzeichner das Ziel AWS-Region als Resource hinzu. Diese Unterzeichnerrolle wird verwendet, um die S3-URL für das Softwareupdate vorab zu signieren. AWS IoT Greengrass Weitere Informationen finden Sie unter [S3-URL-Signer-Rolle](#).

Der Geräteschatten wird nicht mit der Cloud synchronisiert.

Lösung: Stellen Sie sicher, dass es AWS IoT Greengrass über Berechtigungen `iot:UpdateThingShadow` und `iot:GetThingShadow` Aktionen in der [Greengrass-Servicerolle](#) verfügt. Wenn die Servicerolle die verwaltete Richtlinie `AWSGreengrassResourceAccessRolePolicy` verwendet, sind diese Berechtigungen standardmäßig enthalten.

Siehe [Beheben von Timeout-Problemen während der Schattensynchronisierung](#).

Im Folgenden sind allgemeine IAM-Probleme aufgeführt, die bei der Arbeit mit auftreten können.
AWS IoT Greengrass

Ich bin nicht berechtigt, IAM auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS IoT Greengrass übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS IoT Greengrass auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin Administrator und möchte anderen Zugriff gewähren AWS IoT Greengrass

Um anderen den Zugriff zu ermöglichen AWS IoT Greengrass, müssen Sie den Personen oder Anwendungen, die Zugriff benötigen, die entsprechenden Berechtigungen erteilen. Wenn Sie Personen und Anwendungen verwalten, weisen Sie Benutzern oder Gruppen Berechtigungssätze zu, um deren Zugriffsebene zu definieren. AWS IAM Identity Center Mit Berechtigungssätzen werden automatisch IAM-Richtlinien erstellt und den IAM-Rollen zugewiesen, die der Person oder Anwendung zugeordnet sind. Weitere Informationen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter [Berechtigungssätze](#).

Wenn Sie IAM Identity Center nicht verwenden, müssen Sie IAM-Entitäten (Benutzer oder Rollen) für die Personen oder Anwendungen erstellen, die Zugriff benötigen. Anschließend müssen Sie der Entität eine Richtlinie anfügen, die dieser die korrekten Berechtigungen in AWS IoT Greengrass gewährt. Nachdem die Berechtigungen erteilt wurden, stellen Sie dem Benutzer oder Anwendungsentwickler die Anmeldeinformationen zur Verfügung. Sie werden diese

Anmeldeinformationen für den Zugriff verwenden AWS. Weitere Informationen zum Erstellen von IAM-Benutzern, -Gruppen, -Richtlinien und -Berechtigungen finden Sie im [IAM-Benutzerhandbuch unter IAM-Identitäten sowie Richtlinien und Berechtigungen in IAM](#).

Ich möchte Personen außerhalb von mir den Zugriff auf meine Ressourcen ermöglichen AWS-Konto AWS IoT Greengrass

Sie können eine IAM-Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre AWS Ressourcen zugreifen können. Sie können angeben, welchen Personen vertraut werden soll, so dass diese die Rolle übernehmen können. Weitere Informationen finden Sie unter [Zugriff auf einen IAM-Benutzer in einem anderen AWS-Konto, den Sie besitzen](#), und [Bereitstellen des Zugriffs auf Amazon Web Services Services-Konten, die Dritten gehören](#), im IAM-Benutzerhandbuch.

AWS IoT Greengrass unterstützt keinen kontoübergreifenden Zugriff auf der Grundlage ressourcenbasierter Richtlinien oder Zugriffskontrolllisten (). ACLs

Konformitätsvalidierung für AWS IoT Greengrass

Informationen darüber, ob AWS-Service ein [AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt](#), finden Sie unter [Umfang nach Compliance-Programm AWS-Services unter](#). Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter [AWS Compliance-Programme AWS](#).

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter [Berichte herunterladen unter](#).

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der [AWS Sicherheitsdokumentation](#).

Resilienz in AWS IoT Greengrass

Die AWS globale Infrastruktur basiert auf den Regionen und Availability Zones von Amazon Web Services. Jede AWS-Region bietet mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken

erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu den Regionen und Availability Zones von Amazon Web Services finden Sie unter [AWS Globale Infrastruktur](#).

Zusätzlich zur AWS globalen Infrastruktur AWS IoT Greengrass bietet es mehrere Funktionen, die Sie bei Ihren Anforderungen an Datenstabilität und Datensicherung unterstützen.

- Wenn der Core die Internetverbindung verliert, können die Client-Geräte weiterhin über das lokale Netzwerk kommunizieren.
- Sie können den Core so konfigurieren, dass unverarbeitete Nachrichten, die für AWS Cloud Ziele bestimmt sind, in einem lokalen Speichercache statt im Arbeitsspeicher gespeichert werden. Der lokale Speichercache kann auch nach einem Neustart des Kerns bestehen bleiben (z. B. nach einer Gruppenbereitstellung oder einem Geräteneustart), sodass er weiterhin Nachrichten verarbeiten AWS IoT Greengrass kann, die für bestimmt sind. AWS IoT Core Weitere Informationen finden Sie unter [the section called “MQTT-Nachrichtenwarteschlange”](#).
- Sie können den Core so konfigurieren, dass eine persistente Sitzung mit dem AWS IoT Core Message Broker eingerichtet wird. Dadurch kann der Kern Nachrichten empfangen, die gesendet werden, während der Kern offline ist. Weitere Informationen finden Sie unter [the section called “Persistente MQTT-Sitzungen mit AWS IoT Core”](#).
- Sie können eine Greengrass-Gruppe so konfigurieren, dass sie Logs in das lokale Dateisystem und in CloudWatch Logs schreibt. Wenn der Kern seine Konnektivität verliert, kann die lokale Protokollierung fortgesetzt werden, CloudWatch -Protokolle werden jedoch mit einer begrenzten Anzahl von Wiederholungsversuchen gesendet. Wenn die Zahl der zulässigen Wiederholungen erreicht ist, wird das Ereignis entfernt. Sie sollten sich auch der [Protokollierungsbeschränkungen](#) bewusst sein.
- Sie können Lambda-Funktionen erstellen, die [Stream-Manager-Streams](#) lesen und die Daten an lokale Speicherziele senden.

Infrastruktursicherheit in AWS IoT Greengrass

Als verwalteter Dienst AWS IoT Greengrass ist er durch AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter [AWS Cloud-Sicherheit](#). Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung

der bewährten Methoden für die Infrastruktursicherheit finden Sie unter [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe für den Zugriff AWS IoT Greengrass über das Netzwerk. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

In einer AWS IoT Greengrass Umgebung verwenden Geräte X.509-Zertifikate und kryptografische Schlüssel, um sich mit dem zu verbinden und zu authentifizieren. AWS Cloud Weitere Informationen finden Sie unter [the section called “Geräteauthentifizierung und -autorisierung”](#).

Konfiguration und Schwachstellenanalyse in AWS IoT Greengrass

IoT-Umgebungen können aus einer großen Anzahl von Geräten mit unterschiedlichsten Funktionen bestehen und sind langlebig und geografisch verteilt. Aufgrund dieser Merkmale ist die Geräteeinrichtung komplex und fehleranfällig. Und da Geräte bezüglich Rechenleistung, Arbeitsspeicher und Speicherkapazitäten eingeschränkt sind, können Verschlüsselung und andere Formen der Sicherheit auf den Geräten selbst nur limitiert eingesetzt werden. Außerdem verwenden Geräte häufig Software mit bekannten Schwachstellen. Diese Faktoren machen IoT-Geräte zu einem attraktiven Ziel für Hacker und erschweren die kontinuierliche Sicherung.

AWS IoT Device Defender begegnet diesen Herausforderungen durch die Bereitstellung von Tools zur Identifizierung von Sicherheitsproblemen und Abweichungen von bewährten Verfahren. Sie können AWS IoT Device Defender damit verbundene Geräte analysieren, prüfen und überwachen, um ungewöhnliches Verhalten zu erkennen und Sicherheitsrisiken zu minimieren. AWS IoT Device Defender kann Geräte überprüfen, um sicherzustellen, dass sie den bewährten Sicherheitsmethoden entsprechen, und abnormales Verhalten auf Geräten erkennen. Dies ermöglicht es, konsistente Sicherheitsrichtlinien für alle Ihre Geräte durchzusetzen und schnell zu reagieren, wenn Geräte gefährdet sind. AWS IoT Greengrass Generiert in Verbindungen mit AWS IoT Core einen [vorhersehbaren Client IDs](#), den Sie mit AWS IoT Device Defender Funktionen verwenden können. Weitere Informationen finden Sie unter [AWS IoT Device Defender](#) im AWS IoT Core - Entwicklerhandbuch.

In AWS IoT Greengrass Umgebungen sollten Sie sich der folgenden Überlegungen bewusst sein:

- Es liegt in Ihrer Verantwortung, Ihre physischen Geräte, das Dateisystem auf Ihren Geräten und das lokale Netzwerk zu sichern.
- AWS IoT Greengrass erzwingt keine Netzwerkisolierung für benutzerdefinierte Lambda-Funktionen, unabhängig davon, ob sie in einem [Greengrass-Container](#) ausgeführt werden oder nicht. Daher ist es für Lambda-Funktionen möglich, mit jedem anderen Prozess zu kommunizieren, der im System oder außerhalb über das Netzwerk ausgeführt wird.

Wenn Sie die Kontrolle über ein Greengrass-Core-Gerät verlieren und Sie verhindern möchten, dass Client-Geräte Daten an den Core übertragen, gehen Sie wie folgt vor:

1. Entfernen Sie den Greengrass-Kern aus der Greengrass-Gruppe.
2. Rotieren Sie das CA-Gruppenzertifikat. In der AWS IoT Konsole können Sie das CA-Zertifikat auf der Einstellungsseite der Gruppe rotieren lassen. In der AWS IoT Greengrass API können Sie die [CreateGroupCertificateAuthority](#)Aktion verwenden.

Wir empfehlen auch, die vollständige Festplattenverschlüsselung zu verwenden, wenn die Festplatte Ihres Kerngeräts anfällig für Diebstahl ist.

AWS IoT Greengrass und Schnittstellen-VPC-Endpunkte (AWS PrivateLink)

Sie können eine private Verbindung zwischen Ihrer VPC und der AWS IoT Greengrass Steuerungsebene herstellen, indem Sie einen VPC-Schnittstellen-Endpunkt erstellen. Sie können diesen Endpunkt verwenden, um Gruppen, Lambda-Funktionen, Bereitstellungen und andere Ressourcen im AWS IoT Greengrass Service zu verwalten. Schnittstellenendpunkte werden mit einer Technologie betrieben [AWS PrivateLink](#), mit der Sie AWS IoT Greengrass APIs privat ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder AWS Direct Connect-Verbindung zugreifen können. Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen für die Kommunikation. AWS IoT Greengrass APIs Datenverkehr zwischen Ihrer VPC und AWS IoT Greengrass verlässt das Amazon-Netzwerk nicht.

Note

Derzeit können Sie Greengrass-Core-Geräte nicht so konfigurieren, dass sie vollständig innerhalb Ihrer VPC funktionieren.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere [Elastic-Netzwerk-Schnittstellen](#) in Ihren Subnetzen dargestellt.

Weitere Informationen finden Sie unter [Schnittstellen-VPC-Endpunkte \(AWS PrivateLink\)](#) im Amazon-VPC-Benutzerhandbuch.

Themen

- [Überlegungen zu AWS IoT Greengrass VPC-Endpunkten](#)
- [Erstellen Sie einen VPC-Schnittstellen-Endpunkt für den Betrieb auf AWS IoT Greengrass Steuerungsebene](#)
- [Erstellen einer VPC-Endpunkttrichtlinie für AWS IoT Greengrass](#)

Überlegungen zu AWS IoT Greengrass VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für einrichten AWS IoT Greengrass, lesen Sie die [Eigenschaften und Einschränkungen des Schnittstellen-Endpunkts](#) im Amazon VPC-Benutzerhandbuch. Beachten Sie außerdem die folgenden Überlegungen:

- AWS IoT Greengrass unterstützt Aufrufe aller API-Aktionen auf der Kontrollebene von Ihrer VPC aus. Die Steuerungsebene umfasst Operationen wie [CreateDeployment](#) und [StartBulkDeployment](#). Die Steuerungsebene umfasst keine Operationen wie [GetDeployment](#) und [Discover](#), bei denen es sich um Operationen auf der Datenebene handelt.
- VPC-Endpunkte für AWS IoT Greengrass werden derzeit in AWS China Regionen nicht unterstützt.

Erstellen Sie einen VPC-Schnittstellen-Endpunkt für den Betrieb auf AWS IoT Greengrass Steuerungsebene

Sie können einen VPC-Endpunkt für die AWS IoT Greengrass Kontrollebene entweder mit der Amazon VPC-Konsole oder mit AWS Command Line Interface (AWS CLI) erstellen. Weitere Informationen finden Sie unter [Erstellung eines Schnittstellenendpunkts](#) im Benutzerhandbuch für Amazon VPC.

Erstellen Sie einen VPC-Endpunkt für die AWS IoT Greengrass Verwendung des folgenden Dienstnamens:

- `com.amazonaws. region. grünes Gras`

Wenn Sie privates DNS für den Endpunkt aktivieren, können Sie API-Anfragen an die AWS IoT Greengrass Verwendung des Standard-DNS-Namens für die Region stellen, z. B. `greengrass.us-east-1.amazonaws.com`. Die Option für ein privates DNS ist standardmäßig aktiviert.

Weitere Informationen finden Sie unter [Zugriff auf einen Service über einen Schnittstellenendpunkt](#) im Benutzerhandbuch für Amazon VPC.

Erstellen einer VPC-Endpunktrichtlinie für AWS IoT Greengrass

Sie können Ihrem VPC-Endpunkt eine Endpunktrichtlinie hinzufügen, die den Zugriff auf den Betrieb der AWS IoT Greengrass Kontrollebene steuert. Die Richtlinie gibt die folgenden Informationen an:

- Prinzipal, der die Aktionen ausführen kann
- Die Aktionen, die der Prinzipal ausführen kann.
- Die Ressourcen, mit denen der Principal Aktionen ausführen kann.

Weitere Informationen finden Sie unter [Steuerung des Zugriffs auf Services mit VPC-Endpunkten](#) im Amazon-VPC-Benutzerhandbuch.

Example Beispiel: VPC-Endpunktrichtlinie für Aktionen AWS IoT Greengrass

Im Folgenden finden Sie ein Beispiel für eine Endpunktrichtlinie für AWS IoT Greengrass. Wenn diese Richtlinie an einen Endpunkt angehängt ist, gewährt sie allen Prinzipalen auf allen Ressourcen Zugriff auf die aufgelisteten AWS IoT Greengrass Aktionen.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
        "greengrass:CreateDeployment",
        "greengrass:StartBulkDeployment"
      ],
      "Resource": "*"
    }
  ]
}
```

Bewährte Sicherheitsmethoden für AWS IoT Greengrass

Dieses Thema enthält bewährte Sicherheitsmethoden für AWS IoT Greengrass.

Erteilen von Mindestberechtigungen

Folgen Sie dem Prinzip der geringsten Rechte, indem Sie die Mindestanzahl an Berechtigungen in IAM-Rollen verwenden. Beschränken Sie die Verwendung des * Platzhalters für die Resource Eigenschaften Action und in Ihren IAM-Richtlinien. Deklarieren Sie stattdessen, wenn möglich, eine endliche Menge von Aktionen und Ressourcen. Weitere Informationen zu den geringsten Berechtigungen und anderen bewährten Methoden für Richtlinien finden Sie unter [the section called “Best Practices für Richtlinien”](#).

Die bewährte Methode mit den geringsten Rechten gilt auch für AWS IoT Richtlinien, die Sie an Ihre Greengrass-Core- und Client-Geräte anhängen.

Hardcodieren Sie Anmeldeinformationen in Lambda-Funktionen nicht fest

Kodieren Sie Anmeldeinformationen in Ihren benutzerdefinierten Lambda-Funktionen nicht fest. So schützen Sie Ihre Anmeldeinformationen besser:

- Um mit AWS Diensten zu interagieren, definieren Sie Berechtigungen für bestimmte Aktionen und Ressourcen in der [Greengrass-Gruppenrolle](#).
- Verwenden Sie [lokale Secrets](#), um Ihre Anmeldeinformationen zu speichern. Oder, wenn die Funktion das AWS SDK verwendet, verwenden Sie Anmeldeinformationen aus der standardmäßigen Anmeldeinformationsanbieterkette.

Keine Protokollierung sensibler Informationen

Sie sollten die Protokollierung von Anmeldeinformationen und anderen persönlich identifizierbaren Informationen (PII) verhindern. Es wird empfohlen, die folgenden Sicherheitsvorkehrungen zu implementieren, auch wenn für den Zugriff auf lokale Protokolle auf einem Kerngerät Root-Rechte und für den Zugriff auf CloudWatch Protokolle IAM-Berechtigungen erforderlich sind.

- Verwenden Sie keine sensiblen Informationen in MQTT-Themenpfaden.
- Verwenden Sie keine vertraulichen Informationen in Gerätenamen, Typen und Attributen in der AWS IoT Core Registrierung.

- Protokollieren Sie keine vertraulichen Informationen in Ihren benutzerdefinierten Lambda-Funktionen.
- Verwenden Sie keine vertraulichen Informationen in den Namen und in den Ressourcen IDs von Greengrass:
 - Konnektoren
 - Kerne
 - Geräte
 - Funktionen
 - Gruppen
 - Logger
 - Ressourcen (lokal, Machine Learning oder Secrets)
 - Subscriptions (Abonnements)

Erstellen gezielter Abonnements

Abonnements steuern den Informationsfluss in einer Greengrass-Gruppe, indem sie definieren, wie Nachrichten zwischen Diensten, Geräten und Lambda-Funktionen ausgetauscht werden. Um sicherzustellen, dass eine Anwendung nur das tun kann, was sie tun soll, sollten Ihre Abonnements Herausgebern erlauben, Nachrichten nur an bestimmte Themen zu senden, und die Abonnenten darauf beschränken, Nachrichten nur von Themen zu erhalten, die für ihre Funktionalität erforderlich sind.

Synchronisieren der internen Uhr Ihres Geräts

Es ist wichtig, dass Sie eine genaue Uhrzeit auf Ihrem Gerät haben. X.509-Zertifikate haben ein Ablaufdatum und eine Ablaufzeit. Die Uhr auf Ihrem Gerät wird verwendet, um sicherzustellen, dass ein Serverzertifikat noch gültig ist. Geräteuhren können im Laufe der Zeit unpräzise werden, oder die Batterien werden entladen.

Weitere Informationen finden Sie in der bewährten Methode [Synchronisieren der internen Uhr Ihres Geräts](#) im AWS IoT Core -Entwicklerhandbuch.

Verwalten der Geräteauthentifizierung mit dem Greengrass Core

Client-Geräte können [FreeRTOS](#) ausführen oder das [AWS IoT Device SDK](#) oder die [AWS IoT Greengrass Discovery-API verwenden, um Discovery-Informationen](#) abzurufen, die für die

Verbindung und Authentifizierung mit dem Core in derselben Greengrass-Gruppe verwendet werden. Zu den Erkennungsinformationen gehören:

- Konnektivitätsinformationen für den Greengrass-Core, der sich in derselben Greengrass-Gruppe wie das Client-Gerät befindet. Diese Informationen beinhalten die Hostadresse und die Portnummer jedes Endpunkts für das Kerngerät.
- Das CA-Gruppenzertifikat, das zum Signieren des lokalen MQTT-Serverzertifikats verwendet wird. Client-Geräte verwenden das Gruppen-CA-Zertifikat, um das vom Core vorgelegte MQTT-Serverzertifikat zu validieren.

Im Folgenden finden Sie bewährte Methoden für Client-Geräte zur Verwaltung der gegenseitigen Authentifizierung mit einem Greengrass-Core. Diese Methoden können dazu beitragen, Ihr Risiko zu verringern, wenn Ihr Kerngerät gefährdet ist.

Überprüfen Sie das lokale MQTT-Serverzertifikat für jede Verbindung.

Client-Geräte sollten das vom Core vorgelegte MQTT-Serverzertifikat jedes Mal validieren, wenn sie eine Verbindung mit dem Core herstellen. Diese Überprüfung ist die Client-Geräteseite der gegenseitigen Authentifizierung zwischen einem Core-Gerät und Client-Geräten. Client-Geräte müssen in der Lage sein, einen Fehler zu erkennen und die Verbindung zu beenden.

Codieren Sie Erkennungsinformationen nicht fest.

Client-Geräte sollten sich auf Erkennungsoperationen verlassen, um Informationen zur Kernkonnektivität und das Gruppen-CA-Zertifikat abzurufen, auch wenn der Core eine statische IP-Adresse verwendet. Client-Geräte sollten diese Erkennungsinformationen nicht fest codieren.

Aktualisieren Sie die Erkennungsinformationen regelmäßig.

Client-Geräte sollten in regelmäßigen Abständen eine Erkennung durchführen, um die Kernverbindungsinformationen und das Gruppen-CA-Zertifikat zu aktualisieren. Es wird empfohlen, dass Client-Geräte diese Informationen aktualisieren, bevor sie eine Verbindung mit dem Core herstellen. Da kürzere Zeiträume zwischen den Erkennungsvorgängen Ihre potenzielle Gefährdungszeit minimieren können, empfehlen wir, dass die Client-Geräte regelmäßig die Verbindung trennen und erneut verbinden, um das Update auszulösen.

Wenn Sie die Kontrolle über ein Greengrass-Core-Gerät verlieren und Sie verhindern möchten, dass Client-Geräte Daten an den Core übertragen, gehen Sie wie folgt vor:

1. Entfernen Sie den Greengrass-Kern aus der Greengrass-Gruppe.

2. Rotieren Sie das CA-Gruppenzertifikat. In der AWS IoT Konsole können Sie das CA-Zertifikat auf der Einstellungsseite der Gruppe rotieren lassen. In der AWS IoT Greengrass API können Sie die [CreateGroupCertificateAuthority](#)Aktion verwenden.

Wir empfehlen auch, die vollständige Festplattenverschlüsselung zu verwenden, wenn die Festplatte Ihres Kerngeräts anfällig für Diebstahl ist.

Weitere Informationen finden Sie unter [the section called “Geräteauthentifizierung und - autorisierung”](#).

Weitere Informationen finden Sie auch unter

- Bewährte [Sicherheitsmethoden finden Sie AWS IoT Core im AWS IoT](#) Entwicklerhandbuch
- [Zehn goldene Sicherheitsregeln für industrielle IoT-Lösungen](#) im Internet der Dinge im AWS offiziellen Blog

Einloggen und Überwachen AWS IoT Greengrass

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit AWS IoT Greengrass und Leistung Ihrer AWS Lösungen. Sie sollten Überwachungsdaten aus allen Teilen Ihrer AWS Lösung sammeln, damit Sie einen etwaigen Ausfall an mehreren Stellen leichter debuggen können. Bevor Sie mit der Überwachung beginnen AWS IoT Greengrass, sollten Sie einen Überwachungsplan erstellen, der Antworten auf die folgenden Fragen enthält:

- Was sind Ihre Überwachungsziele?
- Welche Ressourcen möchten Sie überwachen?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungs-Tools möchten Sie verwenden?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Überwachungstools

AWS stellt Tools bereit, die Sie zur Überwachung verwenden können AWS IoT Greengrass. Sie können einige dieser Tools für die Überwachung konfigurieren. Einige der Tools erfordern manuelle Eingriffe. Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren.

Sie können die folgenden automatisierten Überwachungstools verwenden, um Probleme zu überwachen AWS IoT Greengrass und zu melden:

- Amazon CloudWatch Logs — Überwachen, speichern und greifen Sie auf Ihre Protokolldateien aus AWS CloudTrail oder anderen Quellen zu. Weitere Informationen finden Sie unter [Überwachung von Protokolldateien](#) im CloudWatch Amazon-Benutzerhandbuch.
- AWS CloudTrail Protokollüberwachung — Teilen Sie Protokolldateien zwischen Konten, überwachen CloudTrail Sie Protokolldateien in Echtzeit, indem Sie sie an CloudWatch Logs senden, schreiben Sie Protokollverarbeitungsanwendungen in Java und stellen Sie sicher, dass sich Ihre Protokolldateien nach der Lieferung von nicht geändert haben CloudTrail. Weitere Informationen finden Sie unter [Arbeiten mit CloudTrail Protokolldateien](#) im AWS CloudTrail Benutzerhandbuch.

- Amazon EventBridge — Verwenden Sie EventBridge Ereignisregeln, um Benachrichtigungen über Statusänderungen Ihrer Greengrass-Gruppenbereitstellungen oder protokollierten API-Aufrufe zu erhalten. CloudTrail Weitere Informationen finden Sie unter [the section called “Abrufen von Bereitstellungsbenachrichtigungen”](#) oder [Was ist Amazon EventBridge?](#) im EventBridge Amazon-Benutzerhandbuch.
- Greengrass-Systemzustandstelemetrie — Abonnieren Sie den Empfang von Telemetriedaten, die vom Greengrass-Core gesendet werden. Weitere Informationen finden Sie unter [the section called “Erfassung von Telemetriedaten zur Systemintegrität”](#).
- Lokaler Integritätscheck — Verwenden Sie den Systemstatus APIs , um einen Überblick über den Status der lokalen AWS IoT Greengrass Prozesse auf dem Kerngerät zu erhalten. Weitere Informationen finden Sie unter [the section called “Aufruf der lokalen Health Check-API”](#).

Weitere Informationen finden Sie auch unter

- [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#)
- [the section called “AWS IoT Greengrass API-Aufrufe protokollieren mit AWS CloudTrail”](#)
- [the section called “Abrufen von Bereitstellungsbenachrichtigungen”](#)

Überwachung mit AWS IoT Greengrass Protokollen

AWS IoT Greengrass besteht aus dem Cloud-Dienst und der AWS IoT Greengrass Core-Software. Die AWS IoT Greengrass Core-Software kann Protokolle auf Amazon CloudWatch und in das lokale Dateisystem Ihres Core-Geräts schreiben. Lambda-Funktionen und -Konnektoren, die auf dem Core ausgeführt werden, können auch CloudWatch Protokolle in Logs und das lokale Dateisystem schreiben. Sie können die Protokolle verwenden, um Ereignisse zu überwachen und Probleme zu beheben. Alle AWS IoT Greengrass Protokolleinträge enthalten einen Zeitstempel, eine Protokollebene und Informationen über das Ereignis. Änderungen an den Protokollierungseinstellungen werden wirksam, nachdem Sie die Gruppe bereitgestellt haben.

Die Protokollierung wird auf Gruppenebene konfiguriert. Schritte zum Konfigurieren der Protokollierung für eine Greengrass-Gruppe finden Sie unter [the section called “Konfigurieren Sie die Protokollierung für AWS IoT Greengrass”](#).

Zugreifen auf Protokolle CloudWatch

Wenn Sie die CloudWatch Protokollierung konfigurieren, können Sie die Protokolle auf der Seite Logs der CloudWatch Amazon-Konsole einsehen. Protokollgruppen für AWS IoT Greengrass Protokolle verwenden die folgenden Benennungskonventionen:

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name  
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

Jede Protokollgruppe enthält Protokollströme, die die folgende Namenskonvention verwenden:

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

Bei der Verwendung von CloudWatch Logs sind die folgenden Überlegungen zu beachten:

- Protokolle werden an CloudWatch Logs mit einer begrenzten Anzahl von Wiederholungsversuchen gesendet, falls keine Internetverbindung besteht. Wenn die Zahl der zulässigen Wiederholungen erreicht ist, wird das Ereignis entfernt.
- Es gelten Einschränkungen für Transaktionen, Arbeitsspeicher und anderes. Weitere Informationen finden Sie unter [the section called “Einschränkungen für die Protokollierung”](#).
- Ihre Greengrass-Gruppenrolle muss das Schreiben AWS IoT Greengrass in CloudWatch Logs zulassen. Zum Erteilen der Berechtigungen [betten Sie die folgende Inline-Richtlinie](#) in Ihre Gruppenrolle ein.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "logs:CreateLogGroup",  
        "logs:CreateLogStream",  
        "logs:PutLogEvents",  
        "logs:DescribeLogStreams"  
      ],  
      "Resource": [  
        "arn:aws:logs:*:*:*"  
      ]  
    }  
  ]  
}
```

```
}  
  }  
  ]  
}
```

Note

Sie können einen detaillierteren Zugriff auf Ihre Protokoll-Ressourcen gewähren. Weitere Informationen finden Sie unter [Verwenden identitätsbasierter Richtlinien \(IAM-Richtlinien\) für CloudWatch Protokolle](#) im CloudWatch Amazon-Benutzerhandbuch.

Die Gruppenrolle ist eine IAM-Rolle, die Sie erstellen und Ihrer Greengrass-Gruppe zuordnen. Sie können die Konsole oder die AWS IoT Greengrass API verwenden, um die Gruppenrolle zu verwalten.

Verwenden der Konsole

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Zielgruppe aus.
3. Wählen Sie Einstellungen anzeigen. Unter Gruppenrolle können Sie die Gruppenrolle anzeigen, zuordnen oder deren Zuordnung aufheben.

Anweisungen zum Anfügen der Gruppenrolle finden Sie unter [Gruppenrolle](#).

Verwenden der CLI

- Verwenden Sie den [get-associated-role](#)Befehl, um die Gruppenrolle zu finden.
- Verwenden Sie den [associate-role-to-group](#)Befehl, um die Gruppenrolle zuzuweisen.
- Verwenden Sie den [disassociate-role-from-group](#)Befehl, um die Gruppenrolle zu entfernen.

Wie Sie die Gruppen-ID abrufen, die in diesen Befehle verwendet wird, erfahren Sie unter [the section called “Abrufen der Gruppen-ID”](#).

Zugreifen auf Dateisystemprotokolle

Wenn Sie die Protokollierung des Dateisystems konfigurieren, werden die Protokolldateien unter **greengrass-root**/ggc/var/log auf dem Kerngerät gespeichert. Im Folgenden finden Sie die übergeordnete Verzeichnisstruktur:

```
greengrass-root/ggc/var/log
```

- crash.log
- system
 - log files for each Greengrass system component
- user
 - **region**
 - **account-id**
 - log files generated by each user-defined Lambda function
 - aws
 - log files generated by each connector

Note

Das Standardverzeichnis von /greengrass ist **greengrass-root**. Wenn ein [Schreibverzeichnis](#) konfiguriert wurde, finden Sie auch die Protokolle dort.

Die folgenden Hinweise gelten für die Verwendung von Dateisystemprotokollen:

- Das Lesen von AWS IoT Greengrass Protokollen im Dateisystem erfordert Root-Rechte.
- AWS IoT Greengrass unterstützt größenabhängige Rotation und automatische Bereinigung, wenn die Menge der Protokolldaten dem konfigurierten Limit nahe kommt.
- Die `crash.log`-Datei ist nur in Dateisystemprotokollen vorhanden. Dieses Protokoll wird nicht in Logs geschrieben. CloudWatch
- Es gelten Einschränkungen im Hinblick auf die Festplattennutzung. Weitere Informationen finden Sie unter [the section called “Einschränkungen für die Protokollierung”](#).

Note

Protokolle für die AWS IoT Greengrass Core-Software v1.0 werden im **greengrass-root**/var/log Verzeichnis gespeichert.

Standardkonfiguration für die Protokollierung

Wenn die Protokollierungseinstellungen nicht explizit konfiguriert wurden, AWS IoT Greengrass wird nach der ersten Gruppenbereitstellung die folgende Standardprotokollierungskonfiguration verwendet.

AWS IoT Greengrass Systemkomponenten

- Typ - FileSystem
- Komponente - GreengrassSystem
- Level - INFO
- Leerzeichen - 128 KB

Benutzerdefinierte Lambda-Funktionen

- Typ - FileSystem
- Komponente - Lambda
- Level - INFO
- Leerzeichen - 128 KB

Note

Vor der ersten Bereitstellung schreiben nur Systemkomponenten Protokolle in das Dateisystem, da keine benutzerdefinierten Lambda-Funktionen bereitgestellt werden.

Konfigurieren Sie die Protokollierung für AWS IoT Greengrass

Sie können die AWS IoT Konsole oder die verwenden [AWS IoT Greengrass APIs](#), um die AWS IoT Greengrass Protokollierung zu konfigurieren.

Note

Um das Schreiben von Protokollen in CloudWatch Logs zu ermöglichen AWS IoT Greengrass , muss Ihre Gruppenrolle die [erforderlichen CloudWatch Logs-Aktionen](#) zulassen.

Konfigurieren der Protokollierung (Konsole)

Sie können die Protokollierung auf der Seite Settings (Einstellungen) der Gruppe konfigurieren.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie die Gruppe, in der Sie die Protokollierung konfigurieren möchten.
3. Wählen Sie auf der Gruppenkonfigurationsseite den Tab Logs aus.
4. Wählen Sie den Speicherort für die Protokollierung wie folgt:
 - Um die CloudWatch Protokollierung zu konfigurieren, wählen Sie für die CloudWatch Protokollkonfiguration die Option Bearbeiten.
 - Um die Dateisystemprotokollierung für die Lokale Protokollkonfiguration zu konfigurieren, wählen Sie Edit (Bearbeiten).

Sie können die Protokollierung für einen einzigen Speicherort oder für beide Speicherorte konfigurieren.

5. Wählen Sie im Konfigurationsmodus „Logs bearbeiten“ die Log-Ebene des Greengrass-Systems oder die Log-Ebene für Benutzer-Lambda-Funktionen aus. Sie können eine Komponente oder beide Komponenten wählen.
6. Wählen Sie die niedrigste Ereignisstufe, die Sie protokollieren möchten. Ereignisse, die unterhalb dieses Grenzwert liegen, werden herausgefiltert und nicht gespeichert.
7. Wählen Sie Speichern. Änderungen werden wirksam, nachdem Sie die Gruppe bereitgestellt haben.

Konfigurieren der Protokollierung (API)

Sie können den AWS IoT Greengrass Logger verwenden, um die Protokollierung APIs programmgesteuert zu konfigurieren. Verwenden Sie z. B. die [CreateLoggerDefinition](#)-Aktion, um eine Logger-Definition zu erstellen, die auf einer [LoggerDefinitionVersion](#)-Nutzlast basiert, und die die folgende Syntax verwendet:

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
```

```
    "Component": "GreengrassSystem|Lambda",
    "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
    "Space": "integer"
  },
  {
    "Id": "string",
    ...
  }
]
```

`LoggerDefinitionVersion` ist ein Array mit einem oder mehreren [Logger](#)-Objekten, die die folgenden Eigenschaften haben:

Id

Eine ID für den Logger.

Type

Der Speichermechanismus für Protokollereignisse. Wenn `AWSCloudWatch` es verwendet wird, werden Protokollereignisse an Logs gesendet CloudWatch . Wenn `FileSystem` verwendet wird, werden die Protokollereignisse im lokalen Dateisystem gespeichert.

Zulässige Werte: `AWSCloudWatch`, `FileSystem`

Component

Die Quelle des Protokollereignisses. Wenn `GreengrassSystem` verwendet wird, werden Ereignisse von Greengrass-Systemkomponenten protokolliert. Bei Verwendung von `Lambda` werden Ereignisse von benutzerdefinierten Lambda-Funktionen protokolliert.

Zulässige Werte: `GreengrassSystem`, `Lambda`

Level

Der Schwellenwert für die Protokollstufe. Protokollereignisse, die unterhalb dieses Grenzwert liegen, werden herausgefiltert und nicht gespeichert.

Gültige Werte: `DEBUG`, `INFO` (empfohlen), `WARN`, `ERROR`, `FATAL`

Space

Die maximale Größe des lokalen Speichers in KB, der zum Speichern von Protokollen verwendet werden soll. Dieses Feld gilt nur, wenn `Type` auf `FileSystem` festgelegt ist.

Konfigurationsbeispiel

Das folgende `LoggerDefinitionVersion`-Beispiel verwendet eine Konfiguration für die Protokollierung, die:

- Aktiviert die Dateisystem `ERROR` - und höhere Protokollierung für AWS IoT Greengrass Systemkomponenten.
- Aktiviert die Dateisystemprotokollierung `INFO` (und höher) für benutzerdefinierte Lambda-Funktionen.
- Aktiviert CloudWatch `INFO` (und höher) die Protokollierung für benutzerdefinierte Lambda-Funktionen.

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
        "Type": "AWSCloudWatch"
      }
    ]
  }
}
```

Nach dem Erstellen einer Logger-Definitionsversion können Sie mit ihrem Versions-ARN eine Gruppenversion erstellen, bevor Sie [die Gruppe bereitstellen](#).

Einschränkungen für die Protokollierung

AWS IoT Greengrass hat die folgenden Einschränkungen bei der Protokollierung.

Transaktionen pro Sekunde

Wenn die Protokollierung aktiviert CloudWatch ist, stapelt die Protokollierungskomponente Protokollereignisse lokal CloudWatch, bevor sie an gesendet werden. Sie können also mit einer Geschwindigkeit von mehr als fünf Anfragen pro Sekunde pro Protokollstream protokollieren.

Arbeitsspeicher

Wenn sie so konfiguriert AWS IoT Greengrass ist, dass sie Protokolle an sendet CloudWatch und eine Lambda-Funktion über einen MB/second längeren Zeitraum mehr als 5 protokolliert, füllt sich die interne Verarbeitungspipeline irgendwann. Der theoretische schlimmste Fall liegt bei 6 MB pro Lambda-Funktion.

Taktversatz

Wenn die Protokollierung aktiviert CloudWatch ist, signiert die Protokollierungskomponente Anfragen, sodass sie den normalen Signaturprozess von Signature Version 4 CloudWatch verwenden. Wenn die Systemzeit auf dem AWS IoT Greengrass Kerngerät um mehr als [15 Minuten](#) nicht mehr synchron ist, werden die Anfragen abgelehnt.

Festplattennutzung

Verwenden Sie die folgende Formel, um die maximale Gesamtgröße der genutzten Festplatte für die Protokollierung zu berechnen.

```
greengrass-system-component-space * 8      // 7 if automatic IP detection is disabled
+ 128KB                                     // the internal log for the local logging
component
+ lambda-space * lambda-count           // different versions of a Lambda function are
treated as one
```

Wobei Folgendes gilt:

greengrass-system-component-space

Die maximale Menge an lokalem Speicher für die Protokolle der AWS IoT Greengrass Systemkomponenten.

lambda-space

Die maximale Menge an lokalem Speicher für Lambda-Funktionsprotokolle.

lambda-count

Die Anzahl der bereitgestellten Lambda-Funktionen.

Protokollverlust

Wenn Ihr AWS IoT Greengrass Kerngerät so konfiguriert ist, dass es nur protokolliert, CloudWatch und es keine Internetverbindung gibt, haben Sie keine Möglichkeit, die Protokolle abzurufen, die sich derzeit im Speicher befinden.

Wenn Lambda-Funktionen beendet werden (z. B. während der Bereitstellung), werden Protokolle für einige Sekunden nicht in Protokolle geschrieben. CloudWatch

CloudTrail Logs

AWS IoT Greengrass läuft mit AWS CloudTrail, einem Dienst, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS IoT Greengrass. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass API-Aufrufe protokollieren mit AWS CloudTrail”](#).

AWS IoT Greengrass API-Aufrufe protokollieren mit AWS CloudTrail

AWS IoT Greengrass ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS IoT Greengrass. CloudTrail erfasst alle API-Aufrufe AWS IoT Greengrass als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS IoT Greengrass Konsole und Codeaufrufen für die AWS IoT Greengrass API-Operationen. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für AWS IoT Greengrass. Wenn Sie keinen Trail konfigurieren,

können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage ermitteln CloudTrail, an die die Anfrage gestellt wurde AWS IoT Greengrass, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details.

Weitere Informationen CloudTrail dazu finden Sie im [AWS CloudTrail Benutzerhandbuch](#).

AWS IoT Greengrass Informationen in CloudTrail

CloudTrail ist auf Ihrem aktiviert AWS-Konto , wenn Sie das Konto erstellen. Wenn eine Aktivität in stattfindet AWS IoT Greengrass, wird diese Aktivität zusammen mit anderen CloudTrail AWS Serviceereignissen im Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem anzeigen, suchen und herunterladen AWS-Konto. Weitere Informationen finden Sie unter [Ereignisse mit CloudTrail Ereignisverlauf anzeigen](#).

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich der Ereignisse für AWS IoT Greengrass, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS-Region s. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- [Übersicht zum Erstellen eines Trails](#)
- [CloudTrail unterstützte Dienste und Integrationen](#)
- [Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail](#)
- [Empfangen von CloudTrail Protokolldateien aus mehreren Regionen](#) und [Empfangen von CloudTrail Protokolldateien von mehreren Konten](#)

Alle AWS IoT Greengrass Aktionen werden von der [AWS IoT Greengrass API-Referenz](#) protokolliert CloudTrail und sind in dieser dokumentiert.

Beispielsweise generieren Aufrufe der CreateFunctionDefinition Aktionen AssociateServiceRoleToAccountGetGroupVersion,GetConnectivityInfo, und Einträge in den CloudTrail Protokolldateien.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter [CloudTrail -Element userIdentity](#).

AWS IoT Greengrass Logdateieinträge verstehen

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `AssociateServiceRoleToAccount` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T17:04:02Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "AssociateServiceRoleToAccount",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "errorCode": "BadRequestException",
  "requestParameters": null,
  "responseElements": {
    "Message": "That role ARN is invalid."
  }
}
```

```

},
"requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
"eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
"readOnly": false,
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}

```

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `GetGroupVersion` Aktion demonstriert.

```

{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-10-17T18:14:57Z"
      }
    }
  },
  "invokedBy": "apimanager.amazonaws.com",
},
"eventTime": "2018-10-17T18:15:11Z",
"eventSource": "greengrass.amazonaws.com",
"eventName": "GetGroupVersion",
"awsRegion": "us-east-1",
"sourceIPAddress": "203.0.113.12",
"userAgent": "apimanager.amazonaws.com",
"requestParameters": {
  "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
  "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
},
"responseElements": null,
"requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
"eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
"readOnly": true,
"eventType": "AwsApiCall",

```

```
"recipientAccountId": "123456789012"
}
```

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `GetConnectivityInfo` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T17:02:12Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "GetConnectivityInfo",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "ThingName": "us-east-1_CIS_1539795000000_"
  },
  "responseElements": null,
  "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
  "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}
```

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `CreateFunctionDefinition` Aktion demonstriert.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/Mary_Major",
```

```

    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2018-10-17T18:01:11Z",
  "eventSource": "greengrass.amazonaws.com",
  "eventName": "CreateFunctionDefinition",
  "awsRegion": "us-east-1",
  "sourceIpAddress": "203.0.113.12",
  "userAgent": "apimanager.amazonaws.com",
  "requestParameters": {
    "InitialVersion": "****"
  },
  "responseElements": {
    "CreationTimestamp": "2018-10-17T18:01:11.449Z",
    "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
    "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-
b983-ee5cfEXAMPLE",
    "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
    "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
    "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/
functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
  },
  "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
  "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
}

```

Weitere Informationen finden Sie auch unter

- [Was ist AWS CloudTrail?](#) im AWS CloudTrail Benutzerhandbuch
- [Erstellen einer EventBridge Regel, die bei einem AWS API-Aufruf ausgelöst wird, mithilfe CloudTrail](#) des EventBridge Amazon-Benutzerhandbuchs
- [AWS IoT Greengrass API-Referenz](#)

Erfassung von Telemetriedaten zur Systemintegrität von AWS IoT Greengrass Kerngeräten

Telemetriedaten zur Systemintegrität sind Diagnosedaten, die Ihnen helfen können, die Leistung kritischer Operationen auf Ihren Greengrass-Kerngeräten zu überwachen. Der Telemetrieagent auf dem Greengrass Core sammelt lokale Telemetriedaten und veröffentlicht sie an Amazon, EventBridge ohne dass eine Kundeninteraktion erforderlich ist. Core-Geräte veröffentlichen Telemetriedaten nach bestem Wissen und EventBridge Gewissen. Beispielsweise können Core-Geräte Telemetriedaten möglicherweise nicht liefern, wenn sie offline sind.

Note

Amazon EventBridge ist ein Event-Bus-Service, mit dem Sie Ihre Anwendungen mit Daten aus einer Vielzahl von Quellen verbinden können, wie z. B. Greengrass-Core-Geräten und [Bereitstellungsbenachrichtigungen](#). Weitere Informationen finden Sie unter [Was ist Amazon EventBridge?](#) im EventBridge Amazon-Benutzerhandbuch.

Sie können Projekte und Anwendungen erstellen, um Telemetriedaten von Ihren Edge-Geräten abzurufen, zu analysieren, zu transformieren und zu melden. Branchenexperten, wie z. B. Verfahrenstechniker, können diese Anwendungen nutzen, um Einblicke in den Zustand der Flotte zu gewinnen.

Um sicherzustellen, dass die Greengrass Edge-Komponenten ordnungsgemäß funktionieren, AWS IoT Greengrass verwendet die Daten für Entwicklungs- und Qualitätsverbesserungszwecke. Diese Funktion hilft auch bei der Entwicklung neuer und verbesserter Edge-Funktionen. AWS IoT Greengrass speichert Telemetriedaten nur für bis zu sieben Tage.

Diese Funktion ist in der AWS IoT Greengrass Core-Software v1.11.0 verfügbar und standardmäßig für alle Greengrass-Kerne aktiviert, einschließlich vorhandener Kerne. Sie beginnen automatisch mit dem Empfang von Daten, sobald Sie auf die AWS IoT Greengrass Core-Software v1.11.0 oder höher aktualisieren.

Informationen darüber, wie Sie auf veröffentlichte Telemetriedaten zugreifen oder diese verwalten können, finden Sie unter [the section called “Abonnieren Sie den Empfang von Telemetriedaten”](#)

Der Telemetrieagent sammelt und veröffentlicht die folgenden Systemmetriken.

Telemetrie-Metriken

Name	Description	Quelle
SystemMemUsage	Die Menge an Speicher, die derzeit von allen Anwendungen auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebssystems.	System (System)
CpuUsage	Die Menge an CPU, die derzeit von allen Anwendungen auf dem Greengrass-Core-Gerät verwendet wird, einschließlich des Betriebssystems.	System (System)
TotalNumberOfFDs	Die Anzahl der Dateideskriptoren, die vom Betriebssystem des Greengrass-Core-Geräts gespeichert werden. Ein Dateideskriptor identifiziert eindeutig eine geöffnete Datei.	System (System)
LambdaOutOfMemory	Die Anzahl der Läufe, die dazu führen, dass der Lambda-Funktion der Speicher ausgeht.	System (System)
DroppedMessageCount	Die Anzahl der verworfenen Nachrichten, die bestimmt sind für. AWS IoT Core	GGCloudSpooler Systemkomponente
LambdaTimeout	Die Anzahl der Timeouts für die Ausführung der benutzerdefinierten Lambda-Funktion.	Benutzerdefinierte Lambda-Funktion und AWS Cloud System

Name	Description	Quelle
LambdaUngracefully Killed	Die Anzahl der Läufe, die die benutzerdefinierte Lambda-Funktion nicht abschließen kann.	Benutzerdefinierte Lambda-Funktion und AWS Cloud System
LambdaError	Die Anzahl der Läufe, die dazu führen, dass die benutzerdefinierte Lambda-Funktion Fehlerprotokolle schreibt.	Benutzerdefinierte Lambda-Funktion und AWS Cloud System
BytesAppended	Die Anzahl der Datenbytes, die an den Stream-Manager angehängt wurden.	GGStreamManager Systemkomponente
BytesUploadedToIoT Analytics	Die Anzahl der Datenbytes, die der Stream-Manager in Kanäle exportiert AWS IoT Analytics.	GGStreamManager Systemkomponente
BytesUploadedToKinesis	Die Anzahl der Datenbytes, die Stream Manager in Streams in Amazon Kinesis Data Streams exportiert.	GGStreamManager Systemkomponente
BytesUploadedToIoT SiteWise	Die Anzahl der Datenbytes, in denen der Stream Manager in Asset-Eigenschaften exportiert AWS IoT SiteWise.	GGStreamManager Systemkomponente
BytesUploadedToS3ExportTaskExecutor	Die Anzahl der Datenbytes, die der Stream-Manager in Objekte in Amazon S3 exportiert.	GGStreamManager Systemkomponente
BytesUploadedToHTTP	Die Anzahl der Datenbytes, die der Stream-Manager nach HTTP exportiert.	GGStreamManager Systemkomponente

Konfiguration der Telemetreeinstellungen

Die Greengrass-Telemetrie verwendet die folgenden Einstellungen:

- Der Telemetrieagent aggregiert jede Stunde Telemetriedaten.
- Der Telemetrieagent veröffentlicht alle 24 Stunden eine Telemetriemeldung.

Note

Die Einstellungen sind unveränderbar.

Sie können die Telemetriefunktion für ein Greengrass-Core-Gerät aktivieren oder deaktivieren. AWS IoT Greengrass verwendet [Schatten](#), um die Telemetriedatenkonfiguration zu verwalten. Ihre Änderungen werden sofort wirksam, wenn der Core eine Verbindung zu AWS IoT Core hat.

Der Telemetrieagent veröffentlicht Daten mithilfe des MQTT-Protokolls mit einer Dienstgütestufe (QoS) von 0. Das bedeutet, dass er weder die Zustellung bestätigt noch versucht, die Veröffentlichung zu wiederholen. Telemetrienachrichten teilen sich eine MQTT-Verbindung mit anderen Nachrichten für Abonnements, für die sie bestimmt sind. AWS IoT Core

Abgesehen von Ihren Datenverbindungskosten AWS IoT Core ist die Datenübertragung vom Core zum Core kostenlos. Das liegt daran, dass der Agent zu einem AWS reservierten Thema veröffentlicht. Abhängig von Ihrem Anwendungsfall können jedoch Kosten anfallen, wenn Sie die Daten erhalten oder verarbeiten.

Voraussetzungen

Bei der Konfiguration der Telemetreeinstellungen gelten die folgenden Anforderungen:

- Sie müssen die AWS IoT Greengrass Core-Software v1.11.0 oder höher verwenden.

Note

Wenn Sie eine frühere Version verwenden und keine Telemetrie verwenden möchten, müssen Sie nichts tun.

- Sie müssen IAM-Berechtigungen bereitstellen, um den Core (Thing) -Shadow zu aktualisieren und die Konfiguration aufzurufen, APIs bevor Sie die Telemetreeinstellungen aktualisieren.

Mit der folgenden Beispiel-IAM-Richtlinie können Sie die Shadow- und Runtime-Konfiguration eines bestimmten Kerns verwalten:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowManageShadow",
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot>DeleteThingShadow",
        "iot:DescribeThing"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
      ]
    },
    {
      "Sid": "AllowManageRuntimeConfig",
      "Effect": "Allow",
      "Action": [
        "greengrass:GetCoreRuntimeConfiguration",
        "greengrass:UpdateCoreRuntimeConfiguration"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:thing/core-name"
      ]
    }
  ]
}
```

Sie können granularen oder bedingten Zugriff auf Ressourcen gewähren, indem Sie beispielsweise ein * Platzhalter-Benennungsschema verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Hinzufügen und Entfernen von IAM-Richtlinien](#).

Konfigurieren Sie die Telemetrieinstellungen (Konsole)

Im Folgenden wird gezeigt, wie Sie die Telemetrieinstellungen eines Greengrass-Cores in der AWS IoT Greengrass Konsole aktualisieren.

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie unter Greengrass-Gruppen Ihre Zielgruppe aus.
3. Wählen Sie auf der Gruppenkonfigurationsseite im Abschnitt Übersicht Ihren Greengrass-Core aus.
4. Wählen Sie auf der Konfigurationsseite des Cores die Registerkarte Telemetrie aus.
5. Wählen Sie im Abschnitt Systemintegritätstelemetrie die Option Konfigurieren aus.
6. Wählen Sie unter Telemetrie konfigurieren die Option Telemetrie aus, um den Telemetriestatus zu aktivieren oder zu deaktivieren.

Important

Standardmäßig ist die Telemetriefunktion für die AWS IoT Greengrass Core-Software v1.11.0 oder höher aktiviert.

Die Änderungen werden zur Laufzeit wirksam. Sie müssen die Gruppe nicht bereitstellen.

Telemetrieinstellungen (CLI) konfigurieren

In der AWS IoT Greengrass API repräsentiert das `TelemetryConfiguration` Objekt die Telemetrieinstellungen eines Greengrass-Kerns. Dieses Objekt ist Teil des `RuntimeConfiguration` Objekts, das dem Kern zugeordnet ist. Sie können die AWS IoT Greengrass API oder das AWS SDK verwenden AWS CLI, um die Greengrass-Telemetrie zu verwalten. Für die Beispiele in diesem Abschnitt wird die AWS CLI verwendet.

Um die Telemetrieinstellungen zu überprüfen

Der folgende Befehl ruft die Telemetrieinstellungen eines Greengrass-Kerns ab.

- Durch *core-thing-name* den Namen des Zielkerns ersetzen.

Um den Namen der Sache zu erhalten, verwenden Sie den [get-core-definition-version](#) Befehl. Der Befehl gibt den ARN des Dings zurück, das den Dingnamen enthält.

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

Der Befehl gibt ein `GetCoreRuntimeConfigurationResponse` Objekt in der JSON-Antwort zurück. Beispiel:

```
{
  "RuntimeConfiguration": {
    "TelemetryConfiguration": {
      "ConfigurationSyncStatus": "OutOfSync",
      "Telemetry": "On"
    }
  }
}
```

Um Telemetreeinstellungen zu konfigurieren

Der folgende Befehl aktualisiert die Telemetreeinstellungen für einen Greengrass-Core.

- Ersetzen Sie *core-thing-name* durch den Namen des Zielkerns.

Um den Namen der Sache zu erhalten, verwenden Sie den [get-core-definition-version](#) Befehl. Der Befehl gibt den ARN des Dings zurück, das den Dingnamen enthält.

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration '{
  "RuntimeConfiguration": {
    "TelemetryConfiguration": {
      "ConfigurationSyncStatus": "InSync",
      "Telemetry": "Off"
    }
  }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus\":"
\": \"InSync\", \"Telemetry\": \"Off\"}}"
```

Änderungen an den Telemetrieinstellungen wurden vorgenommen, falls dies der Fall `ConfigurationSyncStatus` ist `InSync`. Die Änderungen werden zur Laufzeit wirksam. Sie müssen die Gruppe nicht bereitstellen.

TelemetryConfiguration Objekt

Das `TelemetryConfiguration` Objekt hat die folgenden Eigenschaften:

`ConfigurationSyncStatus`

Überprüft, ob die Telemetrieinstellungen synchron sind. Möglicherweise nehmen Sie keine Änderungen an dieser Eigenschaft vor.

Type: Zeichenkette

Gültige Werte: `InSync` oder `OutOfSync`.

Telemetry

Schaltet die Telemetrie ein oder aus. Der Standardwert ist `On`.

Type: Zeichenkette

Gültige Werte: `On` oder `Off`.

Abonnieren Sie den Empfang von Telemetriedaten

Sie können in Amazon Regeln erstellen `EventBridge`, die definieren, wie Telemetriedaten verarbeitet werden, die vom Greengrass-Core-Gerät veröffentlicht wurden. Wenn es die Daten `EventBridge` empfängt, ruft es die in Ihren Regeln definierten Zielaktionen auf. Sie können beispielsweise Ereignisregeln erstellen, die Benachrichtigungen senden, Ereignisinformationen speichern, Korrekturmaßnahmen ergreifen oder andere Ereignisse auslösen.

Telemetrieereignis

Das Ereignis für eine Änderung des Bereitstellungsstatus, einschließlich der Telemetriedaten, verwendet das folgende Format:

```
{
  "version": "0",
  "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
```



```
"detail-type": "Greengrass Telemetry Data",
"source": "aws.greengrass",
"account": "123456789012",
"time": "2020-07-28T20:45:53Z",
"region": "us-west-1",
"resources": [],
"detail": {
  "ThingName": "CoolThing",
  "Schema": "2020-06-30",
  "ADP": [
    {
      "TS": 123231546,
      "NS": "StreamManager",
      "M": [
        {
          "N": "BytesAppended|BytesUploadedToKinesis",
          "Sum": 11,
          "U": "Bytes"
        }
      ]
    },
    {
      "TS": 123231546,
      "NS": "StreamManager",
      "M": [
        {
          "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
          "Sum": 11,
          "U": "Bytes"
        }
      ]
    },
    {
      "TS": 123231546,
      "NS": "StreamManager",
      "M": [
        {
          "N": "BytesAppended|BytesUploadedToHTTP",
          "Sum": 11,
          "U": "Bytes"
        }
      ]
    }
  ]
}
```

```
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
      {
        "N": "BytesAppended|BytesUploadedToIoTAnalytics",
        "Sum": 11,
        "U": "Bytes"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "StreamManager",
    "M": [
      {
        "N": "BytesAppended|BytesUploadedToIoTSiteWise",
        "Sum": 11,
        "U": "Bytes"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
    "M": [
      {
        "N": "LambdaTimeout",
        "Sum": 15,
        "U": "Count"
      }
    ]
  },
  {
    "TS": 123231546,
    "NS": "CloudSpooler",
    "M": [
      {
        "N": "DroppedMessageCount",
        "Sum": 15,
        "U": "Count"
      }
    ]
  },
  {
```

```

        "TS": 1593727692,
        "NS": "SystemMetrics",
        "M": [
            {
                "N": "SystemMemUsage",
                "Sum": 11.23,
                "U": "Megabytes"
            },
            {
                "N": "CpuUsage",
                "Sum": 35.63,
                "U": "Percent"
            },
            {
                "N": "TotalNumberOfFDs",
                "Sum": 416,
                "U": "Count"
            }
        ]
    },
    {
        "TS": 1593727692,
        "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
        "M": [
            {
                "N": "LambdaOutOfMemory",
                "Sum": 12,
                "U": "Count"
            },
            {
                "N": "LambdaUngracefullyKilled",
                "Sum": 100,
                "U": "Count"
            },
            {
                "N": "LambdaError",
                "Sum": 7,
                "U": "Count"
            }
        ]
    }
]
}

```

```
}
```

Das ADP Array enthält eine Liste von aggregierten Datenpunkten mit den folgenden Eigenschaften:

TS

Erforderlich Der Zeitstempel, zu dem die Daten aggregiert wurden.

NS

Erforderlich Der Namespace des Systems.

M

Erforderlich Die Liste der Metriken. Eine Metrik enthält die folgenden Eigenschaften:

N

Der Name der [Metrik](#).

Sum

Der aggregierte Metrikwert. Der Telemetrieagent fügt dem vorherigen Gesamtwert neue Werte hinzu, sodass die Summe einen ständig steigenden Wert ergibt. Sie können den Zeitstempel verwenden, um den Wert einer bestimmten Aggregation zu ermitteln. Um beispielsweise den letzten aggregierten Wert zu ermitteln, subtrahieren Sie den vorherigen Wert mit Zeitstempel vom letzten Wert mit Zeitstempel.

U

Die Einheit des metrischen Werts.

ThingName

Erforderlich Der Name des Ding-Geräts, auf das Sie abzielen.

Voraussetzungen für die Erstellung von EventBridge Regeln

Bevor Sie eine EventBridge Regel für erstellen AWS IoT Greengrass, sollten Sie Folgendes tun:

- Machen Sie sich mit Ereignissen, Regeln und Zielen in vertraut EventBridge.
- Erstellen und konfigurieren Sie die [Ziele](#), die durch Ihre EventBridge Regeln aufgerufen werden. Regeln können viele Arten von Zielen aufrufen, z. B. Amazon Kinesis Kinesis-Streams, AWS Lambda Funktionen, Amazon SNS SNS-Themen und Amazon SQS SQS-Warteschlangen.

Ihre EventBridge Regel und die zugehörigen Ziele müssen sich dort befinden, AWS-Region wo Sie Ihre Greengrass-Ressourcen erstellt haben. Weitere Informationen finden Sie unter [Dienstendpunkte und Kontingente](#) in der. Allgemeine AWS-Referenz

Weitere Informationen finden Sie unter [Was ist Amazon EventBridge?](#) und [Erste Schritte mit Amazon EventBridge](#) im EventBridge Amazon-Benutzerhandbuch.

Erstellen Sie eine Ereignisregel zum Abrufen von Telemetriedaten (Konsole)

Verwenden Sie die folgenden Schritte, um mithilfe der eine EventBridge Regel AWS-Managementkonsole zu erstellen, die vom Greengrass-Core veröffentlichte Telemetriedaten empfängt. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren. Weitere Informationen finden Sie im EventBridge Amazon-Benutzerhandbuch unter [Erstellen einer EventBridge Regel, die bei einem Ereignis von einer AWS Ressource ausgelöst](#) wird.

1. Öffnen Sie die [EventBridgeAmazon-Konsole](#) und wählen Sie Regel erstellen.
2. Geben Sie unter Name and description (Name und Beschreibung) einen Namen und eine Beschreibung für die Regel ein.
3. Wählen Sie Event-Bus — und aktivieren Sie die Regel für den ausgewählten Event-Bus.
4. Wählen Sie den Regeltyp und dann Regel mit einem Ereignismuster.
5. Wählen Sie Weiter aus.
6. Wählen Sie als Ereignisquelle AWS Ereignisse oder EventBridge Partnerereignisse aus.
7. Wählen Sie für Beispielergebnis die Option AWS Ereignisse und anschließend Greengrass Telemetry Data aus.
8. Treffen Sie unter Ereignismuster die folgenden Auswahlen:
 - a. Als Event source (Ereignisquelle) wählen Sie AWS -Services aus.
 - b. Wählen Sie für den AWS Service Greengrass.
 - c. Wählen Sie als Ereignistyp Greengrass Telemetry Data aus.
9. Wählen Sie Weiter aus.
10. Wählen Sie für Ziel 1 Service aus AWS .
11. Wählen Sie für Wählen Sie ein Ziel die Option SQS-Warteschlange aus.
12. Wählen Sie für Queue Ihre Funktion aus.

Erstellen Sie eine Ereignisregel zum Abrufen von Telemetriedaten (CLI)

Verwenden Sie die folgenden Schritte, um mithilfe der eine EventBridge Regel AWS CLI zu erstellen, die vom Greengrass-Core veröffentlichte Telemetriedaten empfängt. Auf diese Weise können Webserver, E-Mail-Adressen und andere Themenabonnenten auf das Ereignis reagieren.

1. Erstellen Sie die Regel.

- Ersetzen Sie *thing-name* durch den Dingnamen des Kerns.

Um den Namen des Dings zu erhalten, verwenden Sie den [get-core-definition-version](#) Befehl. Der Befehl gibt den ARN des Dings zurück, das den Dingnamen enthält.

```
aws events put-rule \  
  --name TestRule \  
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName\":  
    [\"thing-name\"]}}"
```

Eigenschaften, die aus dem Muster weggelassen werden, werden ignoriert.

2. Fügen Sie das Thema als Regelziel hinzu. Im folgenden Beispiel wird Amazon SQS verwendet, Sie können jedoch auch andere Zieltypen konfigurieren.

- *queue-arn* Ersetzen Sie es durch den ARN Ihrer Amazon SQS SQS-Warteschlange.

```
aws events put-targets \  
  --rule TestRule \  
  --targets "Id"="1", "Arn"="queue-arn"
```

Note

Damit Amazon EventBridge Ihre Zielwarteschlange aufrufen kann, müssen Sie Ihrem Thema eine ressourcenbasierte Richtlinie hinzufügen. Weitere Informationen finden Sie unter [Amazon SQS SQS-Berechtigungen](#) im EventBridge Amazon-Benutzerhandbuch.

Weitere Informationen finden Sie unter [Ereignisse und Ereignismuster EventBridge im EventBridge Amazon-Benutzerhandbuch](#).

Fehlerbehebung bei AWS IoT Greengrass Telemetrie

Verwenden Sie die folgenden Informationen, um Probleme bei der Konfiguration der AWS IoT Greengrass Telemetrie zu beheben.

Fehler: Die Antwort enthält "ConfigurationStatus,:" OutOfSync ", nachdem Sie den `get-thing-runtime-configuration` Befehl ausgeführt haben

Lösungen:

- Der AWS IoT Device Shadow-Dienst benötigt Zeit, um Runtime-Konfigurationsupdates zu verarbeiten und die Updates für das Greengrass-Core-Gerät bereitzustellen. Sie können warten und später überprüfen, ob die Telemetreeinstellungen synchronisiert sind.
- Vergewissern Sie sich, dass Ihr Core-Gerät online ist.
- Aktivieren Sie [Amazon CloudWatch Logs in AWS IoT Core](#), um den Schatten zu überwachen.
- Verwenden Sie [AWS IoT Metriken](#), um Ihr Ding zu überwachen.

Aufruf der lokalen Health Check-API

AWS IoT Greengrass enthält eine lokale HTTP-API, die eine Momentaufnahme des aktuellen Status der lokalen Worker-Prozesse bereitstellt, die von gestartet wurden AWS IoT Greengrass. Dieser Snapshot enthält benutzerdefinierte Lambda-Funktionen und Lambda-Systemfunktionen. System-Lambda-Funktionen sind Teil der AWS IoT Greengrass Core-Software. Sie werden als lokale Arbeitsprozesse auf dem Kerngerät ausgeführt und verwalten Vorgänge wie Nachrichtenweiterleitung, lokale Shadow-Synchronisierung und automatische Erkennung von IP-Adressen.

Die Health Check-API unterstützt die folgenden Anfragen:

- Senden Sie eine GET Anfrage, [um Gesundheitsinformationen für alle Arbeitnehmer](#) zu erhalten.
- Senden Sie eine POST Anfrage, [um Gesundheitsinformationen für bestimmte Arbeitnehmer](#) zu erhalten.

Anfragen werden lokal auf dem Gerät gesendet und erfordern keine Internetverbindung.

Holen Sie sich Gesundheitsinformationen für alle Arbeitnehmer

Senden Sie eine GET Anfrage, um Gesundheitsinformationen über alle laufenden Arbeitnehmer zu erhalten.

- **port** Ersetzen Sie es durch die Portnummer des IPC.

```
GET http://localhost:port/2016-11-01/health/workers
```

port

Die Portnummer des IPC.

Der Wert kann zwischen 1024 und 65535 variieren. Der Standardwert ist 8000.

Um diese Portnummer zu ändern, können Sie die `ggDaemonPort` Eigenschaft in der `config.json` Datei aktualisieren. Weitere Informationen finden Sie unter [AWS IoT Greengrass Kernkonfigurationsdatei](#).

Beispielanforderung

In der folgenden `curl` Beispielanforderung werden Gesundheitsinformationen für alle Mitarbeiter abgerufen.

```
curl http://localhost:8000/2016-11-01/health/workers
```

JSON-Antwort

Diese Anfrage gibt eine Reihe von Objekten mit [Gesundheitsinformationen für Arbeitnehmer](#) zurück.

Beispielantwort

In der folgenden Beispielantwort werden Objekte mit Gesundheitsinformationen für alle Arbeitsprozesse aufgeführt, die von gestartet wurden AWS IoT Greengrass.

```
[
  {
    "FuncArn": "arn:aws:lambda::function:GGShadowService:1",
    "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
    "ProcessId": "1234",
```



```

    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda:::function:GGSecretManager:1",
    "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
    "ProcessId": "9798",
    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
    "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
    "ProcessId": "11837",
    "WorkerState": "Waiting"
  },
  ...
]

```

Ruft Gesundheitsinformationen zu bestimmten Mitarbeitern ab

Senden Sie eine POST Anfrage, um Gesundheitsinformationen zu bestimmten Arbeitnehmern zu erhalten. *port* Ersetzen Sie es durch die Portnummer des IPC. Die Standardeinstellung ist 8000.

```
POST http://localhost:port/2016-11-01/health/workers
```

Beispielanforderung

Mit der folgenden curl Beispielanforderung werden Gesundheitsinformationen für bestimmte Mitarbeiter abgerufen.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

Hier ist ein Beispiel für einen body.json Anfragetext:

```

{
  "FuncArns": [
    "arn:aws:lambda:::function:GGShadowService:1",
    "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
  ]
}

```

Der Hauptteil der Anfrage enthält ein FuncArns Array.

FuncArns

Eine Liste von Amazon-Ressourcennamen (ARNs) für die Lambda-Funktionen, die die Ziel-Worker repräsentieren.

- Geben Sie für benutzerdefinierte Lambda-Funktionen den ARN der aktuell bereitgestellten Version an. Wenn Sie der Gruppe Lambda-Funktionen mithilfe eines Alias-ARN hinzugefügt haben, können Sie die GET-Anfrage verwenden, um alle Worker abzurufen, und dann die Worker auswählen, nach der ARNs Sie abfragen möchten.
- Geben Sie für Lambda-Systemfunktionen den ARN der entsprechenden Lambda-Funktion an. Weitere Informationen finden Sie unter [the section called “Lambda-Funktionen des Systems”](#).

Typ: Zeichenfolge-Array

Mindestlänge: 1

Maximale Länge: Die Gesamtzahl der Worker, die AWS IoT Greengrass auf dem Core-Gerät gestartet wurden.

JSON-Antwort

Diese Anfrage gibt ein `Workers` Array und ein `InvalidArns` Array zurück.

Workers

Eine Liste von Objekten mit Gesundheitsinformationen für die angegebenen Mitarbeiter.

Typ: Anordnung von [Objekten mit Gesundheitsinformationen](#)

InvalidArns

Eine Liste von Funktionen ARNs , die ungültig sind, einschließlich Funktionen ARNs , denen keine Mitarbeiter zugeordnet sind.

Typ: Zeichenfolge-Array

Beispielantwort

In der folgenden Beispielantwort werden [Objekte mit Gesundheitsinformationen](#) für die angegebenen Mitarbeiter aufgeführt.

```
{
```

```
"Workers": [
  {
    "FuncArn": "arn:aws:lambda::function:GGShadowService:1",
    "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
    "ProcessId": "1234",
    "WorkerState": "Waiting"
  },
  {
    "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function:3",
    "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
    "ProcessId": "11837",
    "WorkerState": "Waiting"
  }
],
"InvalidArns" : [
  "some-malformed-arn",
  "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
]
}
```

Diese Anfrage gibt die folgenden Fehler zurück:

400 Ungültige Anfrage

Der Text der Anfrage ist falsch formatiert. Um dieses Problem zu beheben, verwenden Sie das folgende Format und senden Sie die Anfrage erneut:

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

Die Anfrage von 400 überschreitet die maximale Anzahl von Mitarbeitern

Die im FuncArns Array ARNs angegebene Anzahl von Mitarbeitern übersteigt die Anzahl der Arbeiter.

Informationen zur Gesundheit der Arbeitnehmer

Ein Objekt mit Gesundheitsinformationen enthält die folgenden Eigenschaften:

FuncArn

Der ARN der Lambda-Systemfunktion, die den Worker repräsentiert.

Typ: string

WorkerId

Die ID des Arbeiters. Diese Eigenschaft kann beim Debuggen nützlich sein. Die `runtime.log` Datei und die Lambda-Funktionsprotokolle enthalten die Worker-ID, sodass diese Eigenschaft besonders nützlich sein kann, um eine On-Demand-Lambda-Funktion zu debuggen, die mehrere Instanzen startet.

Typ: string

ProcessId

Die Prozess-ID (PID) des Worker-Prozesses.

Typ: int

WorkerState

Der Status des Arbeiters.

Typ: string

Folgende Staaten sind möglich:

Working

Eine Nachricht wird verarbeitet.

Waiting

Ich warte auf eine Nachricht. Gilt für langlebige Lambda-Funktionen, die als Daemon oder eigenständiger Prozess ausgeführt werden.

Starting

Hochgefahren, losgelegt.

FailedInitialization

Fehler beim Initialisieren.

Terminated

Vom Greengrass-Daemon gestoppt

NotStarted

Der Start ist fehlgeschlagen. Es wurde ein weiterer Startversuch unternommen.

Initialized

Erfolgreich initialisiert.

Lambda-Funktionen des Systems

Sie können Gesundheitsinformationen für die folgenden Lambda-Systemfunktionen anfordern:

GGCloudSpooler

Verwaltet die Warteschlange für MQTT-Nachrichten, die AWS IoT Core als Quelle oder Ziel dienen.

ARN: `arn:aws:lambda:::function:GGCloudSpooler:1`

GGConnManager

Leitet MQTT-Nachrichten zwischen dem Greengrass-Core und den Client-Geräten weiter.

ARN: `arn:aws:lambda:::function:GGConnManager`

GGDeviceCertificateManager

Überwacht den AWS IoT Shadow auf Änderungen an den IP-Endpunkten des Cores und generiert das serverseitige Zertifikat, das von GGConn Manager für die gegenseitige Authentifizierung verwendet wird.

ARN: `arn:aws:lambda:::function:GGDeviceCertificateManager`

GGIPDetector

Verwaltet die automatische IP-Adresserkennung, die es Geräten in der Greengrass-Gruppe ermöglicht, das Greengrass-Core-Gerät zu erkennen. Dieser Dienst ist nicht verfügbar, wenn Sie IP-Adressen manuell angeben.

ARN: `arn:aws:lambda:::function:GGIPDetector:1`

GGSecretManager

Verwaltet die sichere Speicherung lokaler Geheimnisse und den Zugriff über benutzerdefinierte Lambda und Konnektoren.

ARN: `arn:aws:lambda:::function:GGSecretManager:1`

GGShadowService

Verwaltet lokale Shadows für Client-Geräte.

ARN: `arn:aws:lambda:::function:GGShadowService`

GGShadowSyncManager

Synchronisiert lokale Shadows mit den AWS Cloud für das Kerngerät und die Client-Geräte, wenn die `syncShadow` Geräteeigenschaft auf `true` eingestellt ist.

ARN: `arn:aws:lambda:::function:GGShadowSyncManager`

GGStreamManager

Verarbeitet Datenströme lokal und führt automatische Exporte in die AWS Cloud durch.

ARN: `arn:aws:lambda:::function:GGStreamManager:1`

GGTES

Der lokale Token-Austauschdienst, der die in der Greengrass-Gruppenrolle definierten IAM-Anmeldeinformationen abrufen, die der lokale Code für den Zugriff auf Dienste verwendet. AWS

ARN: `arn:aws:lambda:::function:GGTES`

Verschlagworten Sie Ihre Ressourcen AWS IoT Greengrass

Mithilfe von Stichwörtern können Sie Ihre AWS IoT Greengrass Gruppen organisieren und verwalten. Sie können mit Tags Gruppen und Massenbereitstellungen und den Cores, Geräten sowie anderen Ressourcen, die Gruppen hinzugefügt werden, Metadaten zuzuweisen. Tags können auch in IAM-Richtlinien verwendet werden, um den bedingten Zugriff auf Ihre Greengrass-Ressourcen zu definieren.

Note

Derzeit werden Greengrass-Ressourcen-Tags für AWS IoT Abrechnungsgruppen oder Kostenverteilungsberichte nicht unterstützt.

Grundlagen zu Tags (Markierungen)

Mithilfe von Tags können Sie Ihre AWS IoT Greengrass Ressourcen beispielsweise nach Zweck, Eigentümer und Umgebung kategorisieren. Wenn viele Ressourcen desselben Typs vorhanden sind, können Sie basierend auf den angefügten Tags schnell bestimmte Ressourcen identifizieren. Ein Tag besteht aus einem Schlüssel und einem optionalen Wert. Beides können Sie bestimmen. Wir empfehlen die Verwendung von Tag-Schlüsseln für den jeweiligen Ressourcentyp. Die Verwendung einheitlicher Tag-Schlüssel vereinfacht das Verwalten der -Ressourcen. Sie können zum Beispiel eine Reihe von Tags für Ihre Gruppen definieren, mit denen Sie den Fabrikstandort Ihrer Core-Geräte verfolgen können. Weitere Informationen finden Sie unter [Tagging-Strategien in AWS](#).

Tagging-Unterstützung in der Konsole AWS IoT

In der AWS IoT Konsole können Sie Tags für Ihre Group Greengrass-Ressourcen erstellen, anzeigen und verwalten. Beachten Sie vor dem Erstellen von Tags Beschränkungen für Tags. Weitere Informationen finden Sie unter [Konventionen für Benennung und Nutzung von Tags](#) in der Allgemeine Amazon Web Services-Referenz.

So weisen Sie Tags beim Erstellen einer Gruppe zu

Sie können einer Gruppe Tags zuweisen, wenn Sie die Gruppe erstellen. Wählen Sie im Abschnitt „Tags“ die Option Neues Tag hinzufügen aus, um die Eingabefelder für das Tagging anzuzeigen.

So zeigen Sie Tags auf der Gruppenkonfigurationsseite an und verwalten sie

Sie können Tags auf der Gruppenkonfigurationsseite anzeigen und verwalten, indem Sie Einstellungen anzeigen wählen. Wählen Sie im Abschnitt „Tags“ für die Gruppe die Option Tags verwalten aus, um Gruppen-Tags hinzuzufügen, zu bearbeiten oder zu entfernen.

Tagging-Unterstützung in der API AWS IoT Greengrass

Sie können die AWS IoT Greengrass API verwenden, um Tags für AWS IoT Greengrass Ressourcen zu erstellen, aufzulisten und zu verwalten, die Tagging unterstützen. Beachten Sie vor dem Erstellen von Tags Beschränkungen für Tags. Weitere Informationen finden Sie unter [Konventionen für Benennung und Nutzung von Tags](#) in der Allgemeine Amazon Web Services-Referenz.

- Wenn Sie Tags während der Ressourcenerstellung hinzufügen möchten, definieren Sie sie in der tags-Eigenschaft der Ressource.
- Mit der TagResource-Aktion können Sie Tags nach der Erstellung einer Ressource hinzuzufügen oder Tag-Werte aktualisieren.
- Sie können Tags mit der UntagResource-Aktion aus einer Ressource entfernen.
- Sie können Tags, die einer Ressource zugeordnet sind, mit der ListTagsForResource-Aktion abrufen oder die Ressource abrufen und sich deren tags-Eigenschaft ansehen.

In der folgenden Tabelle sind Ressourcen aufgeführt, die Sie in der AWS IoT Greengrass API taggen können, sowie die entsprechenden Create Get Aktionen.

Ressource	Create	Get
Group	CreateGroup	GetGroup
ConnectorDefinition	CreateConnectorDefinition	GetConnectorDefinition
CoreDefinition	CreateCoreDefinition	GetCoreDefinition
DeviceDefinition	CreateDeviceDefinition	GetDeviceDefinition

Ressource	Create	Get
FunctionDefinition	<u>CreateFunctionDefinition</u>	<u>GetFunctionDefinition</u>
LoggerDefinition	<u>CreateLoggerDefinition</u>	<u>GetLoggerDefinition</u>
ResourceDefinition	<u>CreateResourceDefinition</u>	<u>GetResourceDefinition</u>
SubscriptionDefinition	<u>CreateSubscriptionDefinition</u>	<u>GetSubscriptionDefinition</u>
BulkDeployment	<u>StartBulkDeployment</u>	<u>GetBulkDeploymentsStatus</u>

Sie können Tags für Ressourcen, die Tagging unterstützen, mit den folgenden Aktionen aufführen und verwalten:

- [TagResource](#). Fügt einer Ressource Tags hinzu. Diese werden auch dazu verwendet, den Wert des Schlüssel-Wert-Paars des Tags zu ändern.
- [ListTagsForResource](#). Listet die Tags für eine Ressource auf.
- [UntagResource](#). Entfernt Tags aus einer Ressource.

Sie können die Tags einer Ressource jederzeit hinzufügen, entfernen oder ändern. Wenn Sie den Wert eines Tag-Schlüssels ändern möchten, fügen Sie der Ressource, die denselben Schlüssel und den neuen Wert definiert, ein Tag hinzu. Der neue Wert überschreibt den alten Wert. Sie können einen Wert zwar auf eine leere Zeichenfolge, jedoch nicht Null festlegen.

Wenn Sie eine Ressource löschen, werden die der Ressource zugeordneten Tags ebenfalls gelöscht.

Note

Verwechseln Sie Ressourcen-Tags nicht mit den Attributen, die Sie AWS IoT Dingen zuweisen können. Greengrass-Kerne sind zwar AWS IoT Dinge, aber die in diesem Thema

beschriebenen Ressourcen-Tags sind an eine Sache angehängt *CoreDefinition*, nicht an die Kernsache.

Verwenden von Tags mit IAM-Richtlinien

In Ihren IAM-Richtlinien können Sie Ressourcen-Tags verwenden, um den Benutzerzugriff und die Benutzerberechtigungen zu kontrollieren. Beispielsweise können Richtlinien Benutzern erlauben, nur die Ressourcen zu erstellen, die einen spezifisches Tag aufweisen. Richtlinien können auch verhindern, dass Benutzer Ressourcen mit bestimmten Tags erstellen oder ändern. Sie können Ressourcen während der Erstellung markieren (dies wird als Markierung bei Erstellung bezeichnet). Dadurch müssen Sie später keine benutzerdefinierten Skripts ausführen. Wenn neue Umgebungen mit Tags gestartet werden, werden die entsprechenden IAM-Berechtigungen automatisch angewendet.

Die folgenden Bedingungskontextschlüssel und -werte können im Condition-Element (auch als Condition-Block bezeichnet) der Richtlinie verwendet werden.

`greengrass:ResourceTag/tag-key: tag-value`

Sie können Benutzeraktionen bei Ressourcen mit bestimmten Tags gestatten oder verweigern.

`aws:RequestTag/tag-key: tag-value`

Sie können verlangen, dass ein bestimmter Tag verwendet wird (oder nicht), wenn Sie API-Anforderungen zum Erstellen oder Ändern von Tags bei einer markierbaren Ressource durchführen.

`aws:TagKeys: [tag-key, ...]`

Sie können verlangen, dass ein bestimmter Satz mit Tag-Schlüsseln verwendet wird (oder nicht), wenn eine API-Anforderung zum Erstellen oder Ändern einer markierbaren Ressource durchgeführt wird.

Schlüssel und Werte für den Bedingungskontext können nur für AWS IoT Greengrass Aktionen verwendet werden, die sich auf eine Ressource beziehen, die mit Tags versehen werden kann. Diese Aktionen nehmen die Ressource als einen erforderlicher Parameter an. Sie können beispielsweise bedingten Zugriff auf `GetGroupVersion` festlegen. Sie können keinen bedingten Zugriff auf `AssociateServiceRoleToAccount` festlegen, da keine markierbare Ressource (z. B. Gruppe, Core-Definition oder Gerätedefinition) in der Anfrage verwiesen wird.

Weitere Informationen finden Sie unter [Steuern des Zugriffs mithilfe von Tags](#) und [Referenz zu IAM-JSON-Richtlinien](#) im IAM-Benutzerhandbuch. Die JSON-Richtlinienreferenz enthält detaillierte Syntax, Beschreibungen und Beispiele der Elemente, Variablen und Bewertungslogik von JSON-Richtlinien in IAM.

IAM-Beispielrichtlinien

In der folgenden Beispielrichtlinie werden Tag-basierte Berechtigungen angewendet, mit denen die Aktionen eines Beta-Benutzers nur auf Beta-Ressourcen eingeschränkt werden.

- Die erste Anweisung ermöglicht es einem IAM-Benutzer, nur auf Ressourcen mit dem Tag `env=beta` zu reagieren.
- Die zweite Anweisung verhindert, dass ein IAM-Benutzer das `env=beta`-Tag aus Ressourcen entfernt. Dadurch wird sichergestellt, dass der Benutzer nicht seinen eigenen Zugriff entfernt.

Note

Wenn Sie den Zugriff auf Ressourcen über Tags steuern, sollten Sie auch die Berechtigungen verwalten, mit denen Benutzer diesen Ressourcen Tags hinzuzufügen oder davon entfernen können. Andernfalls können Benutzer in einigen Fällen möglicherweise Ihre Einschränkungen umgehen und sich Zugriff auf eine Ressource verschaffen, indem sie ihre Tags modifizieren.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "beta"
        }
      }
    }
  ],
}
```

```

    {
      "Effect": "Deny",
      "Action": "greengrass:UntagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "beta"
        }
      }
    }
  ]
}

```

Damit Benutzer eine Markierung bei Erstellung durchführen können, müssen Sie Ihnen die erforderlichen Berechtigungen erteilen. Die folgende Beispielrichtlinie enthält die "aws:RequestTag/env": "beta"-Bedingung für die Aktionen `greengrass:CreateGroup` und `greengrass:TagResource`, mit denen Benutzer nur dann eine Gruppe erstellen können, wenn sie die Gruppe mit `env = beta` markieren. Damit wird erreicht, dass Benutzer neue Gruppen markieren müssen.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:TagResource",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "beta"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:CreateGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {

```

```
    "aws:RequestTag/env": "beta"
  }
}
]
```

Im folgenden Ausschnitt wird dargestellt, wie Sie auch mehrere Tag-Werte für einen Tag-Schlüssel angeben können, indem Sie sie in einer Liste angeben:

```
"StringEquals" : {
  "greengrass:ResourceTag/env" : ["dev", "test"]
}
```

Weitere Informationen finden Sie auch unter

- [Taggen von Ressourcen in der AWSAllgemeine Amazon Web Services-Referenz](#)

CloudFormation Unterstützung für AWS IoT Greengrass

CloudFormation ist ein Service, der Sie bei der Erstellung, Verwaltung und Replikation Ihrer AWS Ressourcen unterstützen kann. Sie können CloudFormation Vorlagen verwenden, um AWS IoT Greengrass Gruppen und die Client-Geräte, Abonnements und andere Komponenten zu definieren, die Sie bereitstellen möchten. Ein Beispiel finden Sie unter [the section called “-Beispielvorlage”](#).

Die Ressourcen und Infrastruktur, die Sie aus einer Vorlage generieren, wird als Stack bezeichnet. Sie können alle Ihre Ressourcen in einer Vorlage definieren oder auf Ressourcen von anderen Stacks verweisen. Weitere Informationen zu CloudFormation Vorlagen und Funktionen finden Sie unter [Was ist CloudFormation?](#) im AWS CloudFormation Benutzerhandbuch.

Erstellen von -Ressourcen.

CloudFormation Vorlagen sind JSON- oder YAML-Dokumente, die die Eigenschaften und Beziehungen von AWS Ressourcen beschreiben. Die folgenden AWS IoT Greengrass Ressourcen werden unterstützt:

- Gruppen
- Kerne
- Client-Geräte (Geräte)
- Lambda-Funktionen
- Konnektoren
- Ressourcen (lokal, Machine Learning und geheime Schlüssel)
- Subscriptions (Abonnements)
- Logger (Protokollierungskonfigurationen)

In CloudFormation Vorlagen basieren Struktur und Syntax der Greengrass-Ressourcen auf der AWS IoT Greengrass API. Die [Beispielvorlage verknüpft beispielsweise](#) eine oberste Ebene DeviceDefinition mit einer DeviceDefinitionVersion, die ein einzelnes Client-Gerät enthält. Weitere Informationen finden Sie unter [the section called “Übersicht über das Gruppenobjektmodell”](#).

Die [Referenz zu den AWS IoT Greengrass Ressourcentypen](#) im AWS CloudFormation Benutzerhandbuch beschreibt die Greengrass-Ressourcen, mit CloudFormation denen Sie umgehen können. Wenn Sie CloudFormation Vorlagen verwenden, um Greengrass-Ressourcen zu erstellen,

empfehlen wir, dass Sie diese nur von aus CloudFormation verwalten. Beispielsweise sollten Sie Ihre Vorlage aktualisieren, wenn Sie ein Gerät hinzufügen, ändern oder entfernen möchten (anstatt die AWS IoT Greengrass API oder AWS IoT Konsole zu verwenden). Auf diese Weise können Sie Rollback und andere Funktionen der CloudFormation -Änderungsverwaltung verwenden. Weitere Informationen CloudFormation zur Erstellung und Verwaltung Ihrer Ressourcen und Stacks finden Sie im AWS CloudFormation Benutzerhandbuch unter [Arbeiten mit Stacks](#).

Eine exemplarische Vorgehensweise, die zeigt, wie Sie AWS IoT Greengrass Ressourcen in einer CloudFormation Vorlage erstellen und bereitstellen, finden Sie im offiziellen [Blog unter Automatisieren der AWS IoT Greengrass Einrichtung mit](#) im CloudFormation Internet der Dinge. AWS

Bereitstellen von Ressourcen

Nachdem Sie einen CloudFormation Stack erstellt haben, der Ihre Gruppenversion enthält, können Sie ihn mit der AWS CLI AWS IoT OR-Konsole bereitstellen.

Note

Um eine Gruppe bereitzustellen, muss Ihnen eine Greengrass-Servicerolle zugeordnet sein. AWS-Konto Die Servicerolle ermöglicht AWS IoT Greengrass den Zugriff auf Ihre Ressourcen in AWS Lambda und anderen AWS Diensten. Diese Rolle sollte existieren, wenn Sie in der aktuellen AWS-Region Version bereits eine Greengrass-Gruppe bereitgestellt haben. Weitere Informationen finden Sie unter [the section called “Greengrass-Servicerolle”](#).

So stellen Sie die Gruppe bereit (AWS CLI)

- Führen Sie den Befehl [create-deployment](#) aus.

```
aws greengrass create-deployment --group-id GroupId --group-version-id GroupVersionId --deployment-type NewDeployment
```

Note

Die CommandToDeployGroup Anweisung in der [Beispielvorlage](#) zeigt, wie Sie den Befehl mit Ihrer Gruppen- und Gruppenversion ausgeben, IDs wenn Sie einen Stack erstellen.

So stellen Sie die Gruppe bereit (Konsole)

1. Erweitern Sie im Navigationsbereich der AWS IoT Konsole unter Verwalten die Option Greengrass-Geräte und wählen Sie dann Gruppen (V1) aus.
2. Wählen Sie Ihre Gruppe aus.
3. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Option Bereitstellen aus.

-Beispielvorlage

Die folgende Beispielvorlage erstellt eine Greengrass-Gruppe, die einen Kern, ein Client-Gerät, eine Funktion, einen Logger, ein Abonnement und zwei Ressourcen enthält. Zu diesem Zweck folgt die Vorlage dem Objektmodell der AWS IoT Greengrass API. Beispielsweise sind die Client-Geräte, die Sie der Gruppe hinzufügen möchten, in einer `DeviceDefinitionVersion` Ressource enthalten, die einer `DeviceDefinition` Ressource zugeordnet ist. Wenn Sie der Gruppe die Geräte hinzufügen möchten, verweist die Gruppenversion auf den ARN von `DeviceDefinitionVersion`.

Die Vorlage enthält Parameter, mit denen Sie das Zertifikat ARNs für den Kern und das Gerät sowie den Versions-ARN der Quell-Lambda-Funktion (bei der es sich um eine AWS Lambda Ressource handelt) angeben können. Es verwendet die `GetAtt` systemeigenen Funktionen, um auf, `Ref` und andere Attribute zu verweisen IDs ARNs, die für die Erstellung von Greengrass-Ressourcen erforderlich sind.

Die Vorlage definiert auch zwei AWS IoT Geräte (Dinge), die das Core- und das Client-Gerät darstellen, die der Greengrass-Gruppe hinzugefügt werden.

Nachdem Sie den Stack mit Ihren Greengrass-Ressourcen erstellt haben, können Sie die [Gruppe mithilfe der AWS IoT Konsole AWS CLI oder der Konsole bereitstellen](#).

Note

Die `CommandToDeployGroup`-Anweisung im Beispiel zeigt, wie Sie einen kompletten `create-deployment-CLI`-Befehl ausgeben, mit dem Sie Ihre Gruppe bereitstellen können.

JSON

```
{  
  "AWSTemplateFormatVersion": "2010-09-09",
```



```

    "Description": "AWS IoT Greengrass example template that creates a group version
with a core, device, function, logger, subscription, and resources.",
    "Parameters": {
        "CoreCertificateArn": {
            "Type": "String"
        },
        "DeviceCertificateArn": {
            "Type": "String"
        },
        "LambdaVersionArn": {
            "Type": "String"
        }
    },
    "Resources": {
        "TestCore1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
                "ThingName": "TestCore1"
            }
        },
        "TestCoreDefinition": {
            "Type": "AWS::Greengrass::CoreDefinition",
            "Properties": {
                "Name": "DemoTestCoreDefinition"
            }
        },
        "TestCoreDefinitionVersion": {
            "Type": "AWS::Greengrass::CoreDefinitionVersion",
            "Properties": {
                "CoreDefinitionId": {
                    "Ref": "TestCoreDefinition"
                },
                "Cores": [
                    {
                        "Id": "TestCore1",
                        "CertificateArn": {
                            "Ref": "CoreCertificateArn"
                        },
                        "SyncShadow": "false",
                        "ThingArn": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",

```

```

        {
            "Ref": "AWS::Region"
        },
        {
            "Ref": "AWS::AccountId"
        },
        "thing/TestCore1"
    ]
]
}
}
]
},
"TestClientDevice1": {
    "Type": "AWS::IoT::Thing",
    "Properties": {
        "ThingName": "TestClientDevice1"
    }
},
"TestDeviceDefinition": {
    "Type": "AWS::Greengrass::DeviceDefinition",
    "Properties": {
        "Name": "DemoTestDeviceDefinition"
    }
},
"TestDeviceDefinitionVersion": {
    "Type": "AWS::Greengrass::DeviceDefinitionVersion",
    "Properties": {
        "DeviceDefinitionId": {
            "Fn::GetAtt": [
                "TestDeviceDefinition",
                "Id"
            ]
        },
        "Devices": [
            {
                "Id": "TestClientDevice1",
                "CertificateArn": {
                    "Ref": "DeviceCertificateArn"
                },
                "SyncShadow": "true",
                "ThingArn": {
                    "Fn::Join": [

```

```

        ":" ,
        [
            "arn:aws:iot",
            {
                "Ref": "AWS::Region"
            },
            {
                "Ref": "AWS::AccountId"
            },
            "thing/TestClientDevice1"
        ]
    ]
}
}
}
},
"TestFunctionDefinition": {
    "Type": "AWS::Greengrass::FunctionDefinition",
    "Properties": {
        "Name": "DemoTestFunctionDefinition"
    }
},
"TestFunctionDefinitionVersion": {
    "Type": "AWS::Greengrass::FunctionDefinitionVersion",
    "Properties": {
        "FunctionDefinitionId": {
            "Fn::GetAtt": [
                "TestFunctionDefinition",
                "Id"
            ]
        },
        "DefaultConfig": {
            "Execution": {
                "IsolationMode": "GreengrassContainer"
            }
        },
        "Functions": [
            {
                "Id": "TestLambda1",
                "FunctionArn": {
                    "Ref": "LambdaVersionArn"
                },
                "FunctionConfiguration": {

```

```

        "Pinned": "true",
        "Executable": "run.exe",
        "ExecArgs": "argument1",
        "MemorySize": "512",
        "Timeout": "2000",
        "EncodingType": "binary",
        "Environment": {
            "Variables": {
                "variable1": "value1"
            },
            "ResourceAccessPolicies": [
                {
                    "ResourceId": "ResourceId1",
                    "Permission": "ro"
                },
                {
                    "ResourceId": "ResourceId2",
                    "Permission": "rw"
                }
            ],
            "AccessSysfs": "false",
            "Execution": {
                "IsolationMode": "GreengrassContainer",
                "RunAs": {
                    "Uid": "1",
                    "Gid": "10"
                }
            }
        }
    },
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {

```

```

        "Ref": "TestLoggerDefinition"
    },
    "Loggers": [
        {
            "Id": "TestLogger1",
            "Type": "AWS::CloudWatch",
            "Component": "GreengrassSystem",
            "Level": "INFO"
        }
    ]
},
"TestResourceDefinition": {
    "Type": "AWS::Greengrass::ResourceDefinition",
    "Properties": {
        "Name": "DemoTestResourceDefinition"
    }
},
"TestResourceDefinitionVersion": {
    "Type": "AWS::Greengrass::ResourceDefinitionVersion",
    "Properties": {
        "ResourceDefinitionId": {
            "Ref": "TestResourceDefinition"
        },
        "Resources": [
            {
                "Id": "ResourceId1",
                "Name": "LocalDeviceResource",
                "ResourceDataContainer": {
                    "LocalDeviceResourceData": {
                        "SourcePath": "/dev/TestSourcePath1",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": "false",
                            "GroupOwner": "TestOwner"
                        }
                    }
                }
            },
            {
                "Id": "ResourceId2",
                "Name": "LocalVolumeResourceData",
                "ResourceDataContainer": {
                    "LocalVolumeResourceData": {
                        "SourcePath": "/dev/TestSourcePath2",

```

```

        "DestinationPath": "/volumes/TestDestinationPath2",
        "GroupOwnerSetting": {
            "AutoAddGroupOwner": "false",
            "GroupOwner": "TestOwner"
        }
    }
}
]
}
},
"TestSubscriptionDefinition": {
    "Type": "AWS::Greengrass::SubscriptionDefinition",
    "Properties": {
        "Name": "DemoTestSubscriptionDefinition"
    }
},
"TestSubscriptionDefinitionVersion": {
    "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
    "Properties": {
        "SubscriptionDefinitionId": {
            "Ref": "TestSubscriptionDefinition"
        },
        "Subscriptions": [
            {
                "Id": "TestSubscription1",
                "Source": {
                    "Fn::Join": [
                        ":",
                        [
                            "arn:aws:iot",
                            {
                                "Ref": "AWS::Region"
                            },
                            {
                                "Ref": "AWS::AccountId"
                            },
                            "thing/TestClientDevice1"
                        ]
                    ]
                }
            }
        ],
        "Subject": "TestSubjectUpdated",
        "Target": {
            "Ref": "LambdaVersionArn"
        }
    }
}
}

```

```

    }
  }
]
},
"TestGroup": {
  "Type": "AWS::Greengrass::Group",
  "Properties": {
    "Name": "DemoTestGroupNewName",
    "RoleArn": {
      "Fn::Join": [
        ":",
        [
          "arn:aws:iam:",
          {
            "Ref": "AWS::AccountId"
          },
          "role/TestUser"
        ]
      ]
    },
  },
  "InitialVersion": {
    "CoreDefinitionVersionArn": {
      "Ref": "TestCoreDefinitionVersion"
    },
    "DeviceDefinitionVersionArn": {
      "Ref": "TestDeviceDefinitionVersion"
    },
    "FunctionDefinitionVersionArn": {
      "Ref": "TestFunctionDefinitionVersion"
    },
    "SubscriptionDefinitionVersionArn": {
      "Ref": "TestSubscriptionDefinitionVersion"
    },
    "LoggerDefinitionVersionArn": {
      "Ref": "TestLoggerDefinitionVersion"
    },
    "ResourceDefinitionVersionArn": {
      "Ref": "TestResourceDefinitionVersion"
    }
  },
  "Tags": {
    "KeyName0": "value",
    "KeyName1": "value",

```

```

        "KeyName2": "value"
      }
    }
  },
  "Outputs": {
    "CommandToDeployGroup": {
      "Value": {
        "Fn::Join": [
          " ",
          [
            "groupVersion=$(cut -d'/' -f6 <<<",
            {
              "Fn::GetAtt": [
                "TestGroup",
                "LatestVersionArn"
              ]
            },
            ");",
            "aws --region",
            {
              "Ref": "AWS::Region"
            },
            "greengrass create-deployment --group-id",
            {
              "Ref": "TestGroup"
            },
            "--deployment-type NewDeployment --group-version-id",
            "$groupVersion"
          ]
        ]
      }
    }
  }
}

```

YAML

AWSTemplateFormatVersion: 2010-09-09

Description: >-

AWS IoT Greengrass example template that creates a group version with a core, device, function, logger, subscription, and resources.

Parameters:


```

CoreCertificateArn:
  Type: String
DeviceCertificateArn:
  Type: String
LambdaVersionArn:
  Type: String
Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestCore1
  TestCoreDefinition:
    Type: 'AWS::Greengrass::CoreDefinition'
    Properties:
      Name: DemoTestCoreDefinition
  TestCoreDefinitionVersion:
    Type: 'AWS::Greengrass::CoreDefinitionVersion'
    Properties:
      CoreDefinitionId: !Ref TestCoreDefinition
      Cores:
        - Id: TestCore1
          CertificateArn: !Ref CoreCertificateArn
          SyncShadow: 'false'
          ThingArn: !Join
            - ':'
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestCore1
  TestClientDevice1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestClientDevice1
  TestDeviceDefinition:
    Type: 'AWS::Greengrass::DeviceDefinition'
    Properties:
      Name: DemoTestDeviceDefinition
  TestDeviceDefinitionVersion:
    Type: 'AWS::Greengrass::DeviceDefinitionVersion'
    Properties:
      DeviceDefinitionId: !GetAtt
        - TestDeviceDefinition
        - Id
      Devices:

```

```

- Id: TestClientDevice1
  CertificateArn: !Ref DeviceCertificateArn
  SyncShadow: 'true'
  ThingArn: !Join
    - ':'
    - - 'arn:aws:iot'
      - !Ref 'AWS::Region'
      - !Ref 'AWS::AccountId'
      - thing/TestClientDevice1
TestFunctionDefinition:
  Type: 'AWS::Greengrass::FunctionDefinition'
  Properties:
    Name: DemoTestFunctionDefinition
TestFunctionDefinitionVersion:
  Type: 'AWS::Greengrass::FunctionDefinitionVersion'
  Properties:
    FunctionDefinitionId: !GetAtt
      - TestFunctionDefinition
      - Id
    DefaultConfig:
      Execution:
        IsolationMode: GreengrassContainer
    Functions:
      - Id: TestLambda1
        FunctionArn: !Ref LambdaVersionArn
        FunctionConfiguration:
          Pinned: 'true'
          Executable: run.exe
          ExecArgs: argument1
          MemorySize: '512'
          Timeout: '2000'
          EncodingType: binary
          Environment:
            Variables:
              variable1: value1
            ResourceAccessPolicies:
              - ResourceId: ResourceId1
                Permission: ro
              - ResourceId: ResourceId2
                Permission: rw
          AccessSysfs: 'false'
          Execution:
            IsolationMode: GreengrassContainer
          RunAs:

```

```
        Uid: '1'
        Gid: '10'
TestLoggerDefinition:
  Type: 'AWS::Greengrass::LoggerDefinition'
  Properties:
    Name: DemoTestLoggerDefinition
TestLoggerDefinitionVersion:
  Type: 'AWS::Greengrass::LoggerDefinitionVersion'
  Properties:
    LoggerDefinitionId: !Ref TestLoggerDefinition
    Loggers:
      - Id: TestLogger1
        Type: AWSCloudWatch
        Component: GreengrassSystem
        Level: INFO
TestResourceDefinition:
  Type: 'AWS::Greengrass::ResourceDefinition'
  Properties:
    Name: DemoTestResourceDefinition
TestResourceDefinitionVersion:
  Type: 'AWS::Greengrass::ResourceDefinitionVersion'
  Properties:
    ResourceDefinitionId: !Ref TestResourceDefinition
    Resources:
      - Id: ResourceId1
        Name: LocalDeviceResource
        ResourceDataContainer:
          LocalDeviceResourceData:
            SourcePath: /dev/TestSourcePath1
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
      - Id: ResourceId2
        Name: LocalVolumeResourceData
        ResourceDataContainer:
          LocalVolumeResourceData:
            SourcePath: /dev/TestSourcePath2
            DestinationPath: /volumes/TestDestinationPath2
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
TestSubscriptionDefinition:
  Type: 'AWS::Greengrass::SubscriptionDefinition'
  Properties:
```

```

    Name: DemoTestSubscriptionDefinition
TestSubscriptionDefinitionVersion:
  Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
  Properties:
    SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
    Subscriptions:
      - Id: TestSubscription1
        Source: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
        Subject: TestSubjectUpdated
        Target: !Ref LambdaVersionArn
TestGroup:
  Type: 'AWS::Greengrass::Group'
  Properties:
    Name: DemoTestGroupNewName
    RoleArn: !Join
      - ':'
      - - 'arn:aws:iam:'
        - !Ref 'AWS::AccountId'
        - role/TestUser
    InitialVersion:
      CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
      DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
      FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
      SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
      LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
      ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
    Tags:
      KeyName0: value
      KeyName1: value
      KeyName2: value
Outputs:
  CommandToDeployGroup:
    Value: !Join
      - ' '
      - - groupVersion=$(cut -d'/' -f6 <<<
        - !GetAtt
          - TestGroup
          - LatestVersionArn
        - );

```

```
- aws --region  
- !Ref 'AWS::Region'  
- greengrass create-deployment --group-id  
- !Ref TestGroup  
- '--deployment-type NewDeployment --group-version-id'  
- $groupVersion
```

Unterstützte AWS-Region

Derzeit können Sie AWS IoT Greengrass Ressourcen nur in den folgenden [AWS-Regionen](#) erstellen und verwalten:

- US East (Ohio)
- USA Ost (Nord-Virginia)
- USA West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- Asien-Pazifik (Tokio)
- China (Peking)
- Europe (Frankfurt)
- Europa (Irland)
- Europa (London)
- AWS GovCloud (US-West)

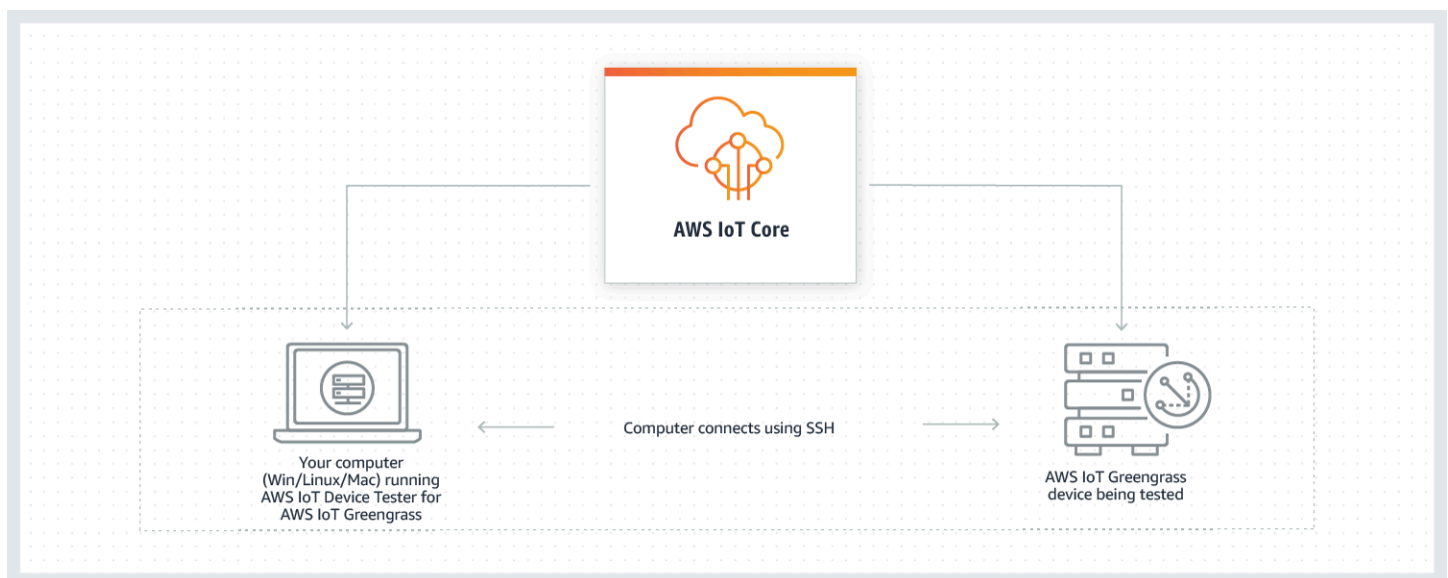
Verwenden von AWS IoT Device Tester für AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) ist ein herunterladbares Test-Framework, mit dem Sie IoT-Geräte validieren können. Da IDT in den [Wartungsmodus](#) versetzt AWS IoT Greengrass Version 1 wurde, generiert es AWS IoT Greengrass V1 keine signierten Qualifikationsberichte mehr. Sie können im Rahmen des [Gerätequalifizierungsprogramms keine neuen AWS IoT Greengrass V1 Geräte mehr für die AWS Partner Aufnahme in den AWS Gerätekatalog qualifizieren](#). Sie können IDT jedoch weiterhin verwenden, um Ihre Greengrass V1-Geräte AWS IoT Greengrass V1 zu testen. Wir empfehlen Ihnen, [IDT for AWS IoT Greengrass V2](#) zu verwenden, um Greengrass-Geräte zu qualifizieren und im [AWS Partner Gerätekatalog](#) aufzulisten.

IDT for AWS IoT Greengrass läuft auf Ihrem Host-Computer (Windows, macOS oder Linux), der mit dem zu testenden Gerät verbunden ist. Er führt Tests durch und fasst die Ergebnisse zusammen. Er bietet auch eine Befehlszeilenschnittstelle zur Verwaltung der Testprozesse.

AWS IoT Greengrass Qualifizierungssuite

Verwenden Sie IDT for, AWS IoT Greengrass um zu überprüfen, ob die AWS IoT Greengrass Core-Software auf Ihrer Hardware läuft und mit der AWS Cloud kommunizieren kann. Es führt auch end-to-end Tests mit AWS IoT Core durch. Es überprüft beispielsweise, ob Ihr Gerät MQTT-Nachrichten senden und empfangen und korrekt verarbeiten kann.



AWS IoT Device Tester für AWS IoT Greengrass organisiert Tests unter Verwendung der Konzepte von Testsuiten und Testgruppen.

- Eine Testsuite ist der Satz von Testgruppen, die verwendet werden, um zu überprüfen, ob ein Gerät mit bestimmten Versionen von AWS IoT Greengrass funktioniert.
- Eine Testgruppe ist ein Satz einzelner Tests, die sich auf eine bestimmte Funktion beziehen, beispielsweise Greengrass-Gruppenbereitstellungen und MQTT-Messaging.

Weitere Informationen finden Sie unter [Verwenden Sie IDT, um die AWS IoT Greengrass Qualification Suite auszuführen](#).

Benutzerdefinierte Testsuiten

Ab IDT v4.0.0 AWS IoT Greengrass kombiniert IDT für ein standardisiertes Konfigurations-Setup und ein standardisiertes Ergebnisformat mit einer Testsuite-Umgebung, mit der Sie benutzerdefinierte Testsuiten für Ihre Geräte und Gerätesoftware entwickeln können. Sie können benutzerdefinierte Tests für Ihre eigene interne Validierung hinzufügen oder sie Ihren Kunden zur Geräteverifizierung zur Verfügung stellen.

Wie ein Testautor eine benutzerdefinierte Testsuite konfiguriert, bestimmt die Einstellungskonfigurationen, die für die Ausführung benutzerdefinierter Testsuiten erforderlich sind. Weitere Informationen finden Sie unter [Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen](#).

Unterstützte Versionen von AWS IoT Device Tester für AWS IoT Greengrass V1

Da IDT in den [Wartungsmodus](#) versetzt AWS IoT Greengrass Version 1 wurde, generiert IDT für AWS IoT Greengrass V1 keine signierten Qualifikationsberichte mehr. Wir empfehlen Ihnen, [IDT](#) für zu verwenden. AWS IoT Greengrass V2

Informationen zu IDT für AWS IoT Greengrass V2 finden Sie unter [Using AWS IoT Device Tester for AWS IoT Greengrass V2](#) im AWS IoT Greengrass V2 Entwicklerhandbuch.

 Note

Sie erhalten eine Benachrichtigung, wenn Sie einen Testlauf starten, falls IDT for nicht mit der Version von kompatibel AWS IoT Greengrass ist, die AWS IoT Greengrass Sie verwenden.

Indem Sie diese Software herunterladen, stimmen Sie der [Lizenzvereinbarung für AWS IoT Device Tester](#) zu.

Nicht unterstützte IDT-Versionen für AWS IoT Greengrass

In diesem Thema werden nicht unterstützte Versionen von IDT for aufgeführt. AWS IoT Greengrass Nicht unterstützte Versionen erhalten keine Fehlerbehebungen oder Updates. Weitere Informationen finden Sie unter [the section called “Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1”](#).

IDT v4.4.1 für die Versionen v1.11.6, v1.10.5 AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die Kernsoftware v1.11.6 und v1.10.5 ausgeführt wird AWS IoT Greengrass .
- Enthält kleinere Fehlerkorrekturen.

Testsuite-Version:

GGQ_1.3.1

- Veröffentlicht 2021.12.20

IDT v4.1.0 für die Versionen v1.11.4, v1.10.4 AWS IoT Greengrass

Versionshinweise:

- Ermöglicht die Validierung und Qualifizierung von Geräten, auf denen die Kernsoftware v1.11.4 und v1.10.4 ausgeführt wird. AWS IoT Greengrass
- Behebt ein Problem, das dazu führte, dass in den Protokollen, die während eines Testlaufs angezeigt wurden, redundante Tags verwendet wurden.

Testsuite-Version:

GGQ_1.3.0

- Veröffentlicht 2021.06.23

- Fügt Wiederholungen für API-Aufrufe zu Lambda und IAM hinzu, AWS STS um die Behandlung von Drosselungen oder Serverproblemen zu verbessern.
- Fügt den ML- und Docker-Testfällen Unterstützung für Python 3.8 hinzu.

IDT v4.0.2 für die AWS IoT Greengrass Versionen v1.11.1, v1.11.0, v1.10.3

Versionshinweise:

- Es wurde ein Problem behoben, das dazu führte, dass IDT Fehler bei der Hardwaresicherheitsintegration (HSI) maskierte.
- Ermöglicht es Ihnen, Ihre benutzerdefinierten Testsuiten mit AWS IoT Device Tester für AWS IoT Greengrass zu entwickeln und auszuführen. Weitere Informationen finden Sie unter [Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen](#).
- Stellt codesignierte IDT-Anwendungen für macOS und Windows bereit. Wenn unter macOS eine Sicherheitswarnung angezeigt wird, müssen Sie möglicherweise eine Sicherheitsausnahme für IDT gewähren. Weitere Informationen finden Sie unter [Sicherheitsausnahme auf macOS](#).

Note

AWS IoT Greengrass stellt kein Dockerfile oder Docker-Image für Version 1.11.1 der Kernsoftware bereit. AWS IoT Greengrass Verwenden Sie eine frühere Version der Kernsoftware, um Ihr Gerät auf die Docker-Qualifikation zu testen. AWS IoT Greengrass

IDT v3.2.0 für die AWS IoT Greengrass Versionen v1.11.0, v1.10.1, v1.10.0

Versionshinweise:

- Standardmäßig führt IDT nur die erforderlichen Qualifizierungstests durch. Um sich für zusätzliche Funktionen zu qualifizieren, können Sie die [device.json](#) Datei ändern.
- Es wurde eine Portnummer hinzugefügt `device.json`, die Sie für SSH-Verbindungen konfigurieren können.
- Docker unterstützt nur [Stream Manager](#) und maschinelles Lernen (ML) ohne Containerisierung. Container, Docker und Hardware Security Integration (HSI) sind für Docker-Geräte nicht verfügbar.
- Wir haben uns zusammengeschlossen `device-ml.json` und zu `device-hsm.json` `device.json`

IDT v3.1.3 für die AWS IoT Greengrass Versionen: v1.10.x, v1.9.x, v1.8.x

Versionshinweise:

- Unterstützung für die Qualifizierung von ML-Funktionen für v1.10.x und v1.9.x hinzugefügt. AWS IoT Greengrass Sie können jetzt mithilfe von IDT überprüfen, ob Ihre Geräte ML-Inferenzen lokal mit Modellen ausführen können, die in der Cloud gespeichert und geschult sind.
- `--stop-on-first-failure` für den Befehl `run-suite` hinzugefügt. Sie können diese Option verwenden, um IDT so zu konfigurieren, dass es beim ersten Fehler nicht mehr läuft. Wir empfehlen, diese Option während der Debugging-Phase auf Testgruppenebene zu verwenden.
- Es wurde eine Takt driftprüfung für MQTT-Tests hinzugefügt, um sicherzustellen, dass das getestete Gerät die richtige Systemzeit verwendet. Die verwendete Zeit muss innerhalb eines akzeptablen Zeitbereichs liegen.
- `--update-idt` für den Befehl `run-suite` hinzugefügt. Sie können diese Option verwenden, um die Antwort für die Aufforderung zum Aktualisieren von IDT festzulegen.
- `--update-managed-policy` für den Befehl `run-suite` hinzugefügt. Mit dieser Option können Sie festlegen, wie auf die Aufforderung zur Aktualisierung der verwalteten Richtlinie reagiert werden soll.
- Es wurde ein Bugfix für automatische Updates von IDT-Testsuite-Versionen hinzugefügt. Der Fix stellt sicher, dass IDT die neuesten Testsuiten ausführen kann, die für Ihre AWS IoT Greengrass Version verfügbar sind.

IDT v3.0.1 für AWS IoT Greengrass

Versionshinweise:

- Unterstützung für v1.10.1 hinzugefügt. AWS IoT Greengrass
- Automatische Updates der IDT-Test-Suite-Versionen. IDT kann die neuesten Testsuiten herunterladen, die für Ihre AWS IoT Greengrass Version verfügbar sind. Mit dieser Funktion gilt:
 - Test-Suites werden in einem *major.minor.patch*-Format versionsgesteuert. Die anfängliche Test-Suite-Version ist GGQ_1.0.0.
 - Sie können neue Test-Suites interaktiv in der Befehlszeilenschnittstelle herunterladen oder beim Starten von IDT das `upgrade-test-suite`-Flag setzen.

Weitere Informationen finden Sie unter [the section called "Test-Suite-Versionen"](#).

- `list-supported-products` hinzugefügt. Mit diesem Befehl können Sie die AWS IoT Greengrass - und Test-Suite-Versionen auflisten, die von der installierten Version von IDT unterstützt werden.
- `list-test-cases` hinzugefügt. Mit diesem Befehl können Sie die Testfälle auflisten, die in einer Testgruppe verfügbar sind.
- `test-id` für den Befehl `run-suite` hinzugefügt. Mit dieser Option können Sie einzelne Testfälle in einer Testgruppe ausführen.

IDT v2.3.0 für AWS IoT Greengrass v1.10, v1.9.x und v1.8.x

Beim Testen auf einem physischen Gerät werden v1.10, v1.9.x und AWS IoT Greengrass v1.8.x unterstützt.

Beim Testen in einem Docker-Container werden v1.10 und v1.9.x unterstützt. AWS IoT Greengrass

Versionshinweise:

- Unterstützung für [the section called “ AWS IoT Greengrass In einem Docker-Container ausführen ”](#) hinzugefügt. Sie können jetzt IDT verwenden, um zu qualifizieren und zu überprüfen, ob Ihre Geräte in einem Docker-Container ausgeführt AWS IoT Greengrass werden können.
- Es wurde eine [AWS verwaltete Richtlinie](#) (`AWSIoTDeviceTesterForGreengrassFullAccess`) hinzugefügt, die die für die Ausführung von AWS IoT Device Tester erforderlichen Berechtigungen definiert. Wenn für neue Versionen zusätzliche Berechtigungen erforderlich sind, AWS fügen Sie diese zu dieser verwalteten Richtlinie hinzu, sodass Sie Ihre IAM-Berechtigungen nicht aktualisieren müssen.
- Es wurden Überprüfungen eingeführt, um zu validieren, ob Ihre Umgebung (z. B. Gerätekonnektivität und Internetkonnektivität) korrekt eingerichtet ist, bevor Sie die Testfälle ausführen.
- Verbesserte Greengrass-Abhängigkeitsprüfung in IDT, um diese flexibler zu machen, während auf Geräten nach `libc` gesucht wird.

IDT v2.2.0 für AWS IoT Greengrass v1.10, v1.9.x und v1.8.x

Versionshinweise:

- Unterstützung für AWS IoT Greengrass v1.10 hinzugefügt.
- Unterstützung für den [Greengrass-Docker-Anwendungsbereitstellungs](#)-Konnektor hinzugefügt.
- Unterstützung für AWS IoT Greengrass [Stream Manager](#) hinzugefügt.
- Unterstützung für die AWS IoT Greengrass Region China (Peking) hinzugefügt.

IDT v2.1.0 für AWS IoT Greengrass v1.9.x, v1.8.x und v1.7.x

Versionshinweise:

- Unterstützung für AWS IoT Greengrass v1.9.4 hinzugefügt.
- Unterstützung für ARMv6l Linux-Geräte hinzugefügt.

IDT v2.0.0 für AWS IoT Greengrass v1.9.3, v1.9.2, v1.9.1, v1.9.0, v1.8.4, v1.8.3 und v1.8.2

Versionshinweise:

- Die Abhängigkeit von Python für das zu testende Gerät wurde entfernt.
- Die Ausführungszeit der Testsuite wurde um mehr als 50 Prozent reduziert, dies bedeutet einen schnelleren Qualifizierungsprozess.
- Die Größe der ausführbaren Datei wurde um mehr als 50 Prozent reduziert, dadurch sind der Download und die Installation schneller.
- Verbesserte [Unterstützung für Timeout-Multiplikatoren](#) für alle Testfälle.
- Verbesserte Meldungen nach der Diagnose, um Fehler schneller zu beheben.
- Die für die Ausführung von IDT erforderliche Berechtigungsrichtlinienvorlage wurde aktualisiert.
- Unterstützung für AWS IoT Greengrass v1.9.3 hinzugefügt.

IDT v1.3.3 für AWS IoT Greengrass v1.9.2, v1.9.1, v1.9.0, v1.8.3 und v1.8.2

Versionshinweise:

- Zusätzliche Unterstützung für Greengrass v1.9.2 und v1.8.3.
- Unterstützung für Greengrass OpenWrt hinzugefügt.
- SSH-Benutzername und Passwort für die Geräteanmeldung hinzugefügt.

- Nativer Test-Bugfix für die OpenWrt ARMv7l Plattform hinzugefügt.

IDT v1.2 für v1.8.1 AWS IoT Greengrass

Versionshinweise:

- Ein konfigurierbarer Timeout-Multiplikator wurde hinzugefügt, um Timeout-Probleme (z.B. bei Verbindungen mit niedriger Bandbreite) zu beheben.

IDT v1.1 für v1.8.0 AWS IoT Greengrass

Versionshinweise:

- Unterstützung für AWS IoT Greengrass Hardware Security Integration (HSI) hinzugefügt.
- Unterstützung für AWS IoT Greengrass Container und keinen Container hinzugefügt.
- Automatisierte Erstellung von AWS IoT Greengrass Servicerollen hinzugefügt.
- Test-Ressourcenbereinigung verbessert.
- Zusammenfassungsbericht für die Testausführung hinzugefügt.

IDT v1.1 für v1.7.1 AWS IoT Greengrass

Versionshinweise:

- Unterstützung für AWS IoT Greengrass Hardware Security Integration (HSI) hinzugefügt.
- Unterstützung für AWS IoT Greengrass Container und keinen Container hinzugefügt.
- Automatisierte Erstellung von AWS IoT Greengrass Servicerollen hinzugefügt.
- Test-Ressourcenbereinigung verbessert.
- Zusammenfassungsbericht für die Testausführung hinzugefügt.

IDT v1.0 für v1.6.1 AWS IoT Greengrass

Versionshinweise:

- OTA-Test-Bugfix für future AWS IoT Greengrass Versionskompatibilität hinzugefügt.

 Note

Wenn Sie IDT v1.0 für AWS IoT Greengrass v1.6.1 verwenden, müssen Sie eine [Greengrass-Servicerolle](#) erstellen. In späteren Versionen erstellt IDT die Servicerolle für Sie.

Verwenden Sie IDT, um die AWS IoT Greengrass Qualification Suite auszuführen

Mit AWS IoT Device Tester (IDT) können Sie überprüfen, ob AWS IoT Greengrass die AWS IoT Greengrass Core-Software auf Ihrer Hardware ausgeführt wird und mit der kommunizieren kann. AWS Cloud Es führt auch end-to-end Tests mit AWS IoT Core durch. Es überprüft beispielsweise, ob Ihr Gerät MQTT-Nachrichten senden und empfangen und korrekt verarbeiten kann.

Da IDT for in den [Wartungsmodus](#) versetzt AWS IoT Greengrass Version 1 wurde, generiert es AWS IoT Greengrass V1 keine signierten Qualifikationsberichte mehr. Wenn Sie Ihre Hardware zum AWS Partner Gerätekatalog hinzufügen möchten, führen Sie die AWS IoT Greengrass V2 Qualifizierungssuite aus, um Testberichte zu erstellen, an die Sie sie senden können. AWS IoT Weitere Informationen finden Sie unter [AWS Gerätequalifizierungsprogramm](#) und [Unterstützte Versionen von IDT für AWS IoT Greengrass V2](#).

Zusätzlich zum Testen von Geräten AWS IoT Greengrass erstellt IDT for Ressourcen (z. B. AWS IoT Dinge, AWS IoT Greengrass Gruppen, Lambda-Funktionen usw.) in Ihrem, um den AWS-Konto Qualifizierungsprozess zu erleichtern.

Um diese Ressourcen zu erstellen, AWS IoT Greengrass verwendet IDT for die in der `config.json` Datei konfigurierten AWS Anmeldeinformationen, um API-Aufrufe in Ihrem Namen durchzuführen. Diese Ressourcen werden zu verschiedenen Zeiten während eines Tests bereitgestellt.

Wenn Sie IDT for verwenden AWS IoT Greengrass , um die AWS IoT Greengrass Qualification Suite auszuführen, führt IDT die folgenden Schritte aus:

1. Lädt und validiert Ihre Geräte- und Anmeldeinformationskonfigurationen.
2. Führen Sie ausgewählte Tests mit den erforderlichen lokalen und Cloud-Ressourcen durch.
3. Bereinigen Sie lokale und Cloud-Ressourcen.
4. Generiert Testberichte, aus denen hervorgeht, ob Ihr Gerät die für die Qualifizierung erforderlichen Tests bestanden hat.

Test-Suite-Versionen

IDT for AWS IoT Greengrass organisiert Tests in Testsuiten und Testgruppen.

- Eine Testsuite ist der Satz von Testgruppen, die verwendet werden, um zu überprüfen, ob ein Gerät mit bestimmten Versionen von AWS IoT Greengrass funktioniert.
- Eine Testgruppe ist ein Satz einzelner Tests, die sich auf eine bestimmte Funktion beziehen, beispielsweise Greengrass-Gruppenbereitstellungen und MQTT-Messaging.

Ab IDT v3.0.0 werden Testsuites beispielsweise in einem *major.minor.patch*-Format versionsgesteuert, beispielsweise GGQ_1.0.0. Wenn Sie IDT herunterladen, enthält das Paket die neueste Test-Suite-Version.

Important

IDT unterstützt die drei neuesten Testsuite-Versionen für die Gerätequalifizierung. Weitere Informationen finden Sie unter [the section called “Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1”](#).

Sie können ausführen `list-supported-products`, um die Versionen AWS IoT Greengrass und Testsuiten aufzulisten, die von Ihrer aktuellen Version von IDT unterstützt werden. Tests von nicht unterstützten Testsuite-Versionen sind für die Gerätequalifizierung nicht gültig. IDT druckt keine Qualifizierungsberichte für nicht unterstützte Versionen.

Aktualisierungen der IDT-Konfigurationseinstellungen

Neue Tests können neue IDT-Konfigurationseinstellungen einführen.

- Wenn die Einstellungen optional sind, führt IDT die Tests weiter aus.
- Wenn die Einstellungen erforderlich sind, benachrichtigt IDT Sie und beendet die Ausführung. Nachdem Sie die Einstellungen konfiguriert haben, starten Sie den Testlauf neu.

Die Konfigurationseinstellungen befinden sich im *<device-tester-extract-location>/configs*-Ordner. Weitere Informationen finden Sie unter [the section called “IDT-Einstellungen konfigurieren”](#).

Wenn eine aktualisierte Testsuite-Version Konfigurationseinstellungen hinzufügt, erstellt IDT eine Kopie der ursprünglichen Konfigurationsdatei in *<device-tester-extract-location>/configs*.

Beschreibung der Testgruppen

IDT v2.0.0 and later

Erforderliche Testgruppen für die Core-Qualifizierung

Diese Testgruppen sind erforderlich, um Ihr AWS IoT Greengrass Gerät für den AWS Partner Gerätekatalog zu qualifizieren.

AWS IoT Greengrass Kernabhängigkeiten

Überprüft, ob Ihr Gerät alle Software- und Hardwareanforderungen für die AWS IoT Greengrass Core-Software erfüllt.

Der `Software Packages Dependencies`-Testfall in dieser Testgruppe ist beim Testen in einem [Docker-Container](#) nicht anwendbar.

Bereitstellung

Überprüft, ob Lambda-Funktionen auf Ihrem Gerät bereitgestellt werden können.

MQTT

Überprüft die Funktionalität des AWS IoT Greengrass Nachrichtenrouters, indem die lokale Kommunikation zwischen dem Greengrass-Core und den Client-Geräten, bei denen es sich um lokale IoT-Geräte handelt, überprüft wird.

Over-the-Air (OTA)

Überprüft, ob Ihr Gerät ein OTA-Update der AWS IoT Greengrass Core-Software erfolgreich durchführen kann.

Diese Testgruppe ist beim Testen in einem [Docker-Container](#) nicht anwendbar.

Version

Überprüft, ob die AWS IoT Greengrass bereitgestellte Version mit der von Ihnen verwendeten AWS IoT Device Tester-Version kompatibel ist.

Optionale Testgruppen

Diese Testgruppen sind optional. Wenn Sie sich für optionale Tests qualifizieren, wird Ihr Gerät mit zusätzlichen Funktionen im AWS Partner Gerätecatalog aufgeführt.

Container-Abhängigkeiten

Überprüft, ob das Gerät alle Software- und Hardwareanforderungen erfüllt, um Lambda-Funktionen im Containermodus auf einem Greengrass-Kern auszuführen.

Diese Testgruppe ist beim Testen in einem [Docker-Container](#) nicht anwendbar.

Bereitstellungscontainer

Überprüft, ob Lambda-Funktionen auf dem Gerät bereitgestellt und im Containermodus auf einem Greengrass-Kern ausgeführt werden können.

Diese Testgruppe ist beim Testen in einem [Docker-Container](#) nicht anwendbar.

Docker-Abhängigkeiten (unterstützt für IDT v2.2.0 und höher)

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um den Greengrass Docker Application Deployment Connector für die Ausführung von Containern zu verwenden

Diese Testgruppe ist beim Testen in einem [Docker-Container](#) nicht anwendbar.

Integration von Hardware-Sicherheit (Hardware Security Integration, HSI)

Überprüft, ob die bereitgestellte gemeinsam genutzte HSI-Bibliothek eine Schnittstelle zum Hardware-Sicherheitsmodul (HSM) herstellen kann, und implementiert das erforderliche PKCS #11 korrekt. APIs Das HSM und die freigegebene Bibliothek müssen in der Lage sein, eine CSR zu signieren, TLS-Operationen auszuführen und die korrekten Schlüssellängen sowie den korrekten Algorithmus für den öffentlichen Schlüssel bereitzustellen.

Stream-Manager-Abhängigkeiten (unterstützt für IDT v2.2.0 und höher)

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um Stream Manager auszuführen. AWS IoT Greengrass

Machine-Learning-Abhängigkeiten (unterstützt für IDT v3.1.0 und höher)

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die ML-Inferenz lokal ausführen zu können.

Machine-Learning-Inferenztests (unterstützt für IDT v3.1.0 und höher)

Überprüft, ob die ML-Inferenz auf dem getesteten Gerät ausgeführt werden kann. Weitere Informationen finden Sie unter [the section called “Optional: Konfigurieren des Geräts für die ML-Qualifizierung”](#).

Machine-Learning-Inferenz-Containertests (unterstützt für IDT v3.1.0 und höher)

Überprüft, ob die ML-Inferenz auf dem getesteten Gerät und im Containermodus auf einem Greengrass Core ausgeführt werden kann. Weitere Informationen finden Sie unter [the section called “Optional: Konfigurieren des Geräts für die ML-Qualifizierung”](#).

IDT v1.3.3 and earlier

Erforderliche Testgruppen für die Core-Qualifizierung

Diese Tests sind erforderlich, um Ihr AWS IoT Greengrass Gerät für den AWS Partner Gerätecatalog zu qualifizieren.

AWS IoT Greengrass Kernabhängigkeiten

Überprüft, ob Ihr Gerät alle Software- und Hardwareanforderungen für die AWS IoT Greengrass Core-Software erfüllt.

Kombination (Gerätesicherheitsinteraktion)

Überprüft die Funktionalität des Gerätezertifikatsmanagers und der IP-Erkennung auf dem Greengrass Core-Gerät, indem die Konnektivitätsinformationen für die Greengrass-Gruppe in der Cloud geändert wird. Die Testgruppe wechselt das AWS IoT Greengrass Serverzertifikat und überprüft, AWS IoT Greengrass ob Verbindungen zugelassen sind.

Bereitstellung (erforderlich für IDT v1.2 und früher)

Überprüft, ob Lambda-Funktionen auf Ihrem Gerät bereitgestellt werden können.

Device Certificate Manager (DCM)

Überprüft, ob der AWS IoT Greengrass Gerätezertifikatsmanager beim Start ein Serverzertifikat generieren und Zertifikate rotieren kann, wenn sie kurz vor Ablauf stehen.

IP-Erkennung (IPD)

Überprüft, ob Informationen zur Core-Verbindung aktualisiert werden, wenn in einem Greengrass Core-Gerät Änderungen von IP-Adressen auftreten. Weitere Informationen finden Sie unter [Aktivieren der automatischen IP-Erkennung](#).

Protokollierung

Überprüft, ob der AWS IoT Greengrass Protokollierungsdienst mithilfe einer in Python geschriebenen Lambda-Benutzerfunktion in eine Protokolldatei schreiben kann.

MQTT

Überprüft die Funktionalität des AWS IoT Greengrass Nachrichtenrouters, indem Nachrichten zu einem Thema gesendet werden, das an zwei Lambda-Funktionen weitergeleitet wird.

Nativ

Überprüft, ob native (kompilierte) Lambda-Funktionen ausgeführt werden AWS IoT Greengrass können.

Over-the-Air (OTA)

Überprüft, ob Ihr Gerät ein OTA-Update der AWS IoT Greengrass Core-Software erfolgreich durchführen kann.

Penetration

Überprüft, ob die AWS IoT Greengrass Core-Software nicht gestartet werden kann, wenn Hardlink/Soft-Link-Schutz und [Seccomp](#) nicht aktiviert sind. Sie wird außerdem verwendet, um weitere sicherheitsrelevante Funktionen zu überprüfen.

Shadow

Überprüft lokale Shadow- und Shadow-Cloud-Synchronisierungsfunktionalität.

Spooler

Prüft, ob die MQTT-Nachrichten mit der Standard-Spooler-Konfiguration in die Warteschlange gestellt werden.

Token Exchange Service (TES)

Überprüft, ob das Kernzertifikat gegen gültige Anmeldeinformationen eingetauscht AWS IoT Greengrass werden kann. AWS

Version

Überprüft, ob die AWS IoT Greengrass bereitgestellte Version von mit der von Ihnen verwendeten AWS IoT Device Tester-Version kompatibel ist.

Optionale Testgruppen

Diese Tests sind optional. Wenn Sie sich für optionale Tests qualifizieren, wird Ihr Gerät mit zusätzlichen Funktionen im AWS Partner Gerätecatalog aufgeführt.

Container-Abhängigkeiten

Überprüft, ob das Gerät alle erforderlichen Abhängigkeiten erfüllt, um Lambda-Funktionen im Container-Modus auszuführen.

Integration von Hardware-Sicherheit (Hardware Security Integration, HSI)

Überprüft, ob die bereitgestellte gemeinsam genutzte HSI-Bibliothek eine Schnittstelle zum Hardware-Sicherheitsmodul (HSM) herstellen kann, und implementiert das erforderliche PKCS #11 korrekt. APIs Das HSM und die freigegebene Bibliothek müssen in der Lage sein, eine CSR zu signieren, TLS-Operationen auszuführen und die korrekten Schlüssellängen sowie den korrekten Algorithmus für den öffentlichen Schlüssel bereitzustellen.

Zugriff auf lokale Ressourcen

Überprüft die LRA-Funktion (Local Resource Access) von, AWS IoT Greengrass indem es den Zugriff auf lokale Dateien und Verzeichnisse, die verschiedenen Linux-Benutzern und -Gruppen gehören, für containerisierte Lambda-Funktionen über LRA ermöglicht. AWS IoT Greengrass APIs Lambda-Funktionen sollte der Zugriff auf lokale Ressourcen basierend auf der Konfiguration des lokalen Ressourcenzugriffs erlaubt oder verweigert werden.

Netzwerk

Überprüft, ob Socket-Verbindungen von einer Lambda-Funktion hergestellt werden können. Diese Socket-Verbindungen sollten basierend auf der Greengrass Core-Konfiguration zugelassen oder verweigert werden.

Voraussetzungen für den Betrieb der AWS IoT Greengrass Qualification Suite

In diesem Abschnitt werden die Voraussetzungen für die Verwendung von AWS IoT Device Tester (IDT) AWS IoT Greengrass zum Ausführen der AWS IoT Greengrass Qualification Suite beschrieben.

Laden Sie die neueste Version von AWS IoT Device Tester herunter für AWS IoT Greengrass

Laden Sie die [neueste Version](#) von IDT herunter und extrahieren Sie die Software an einen Speicherort in Ihrem Dateisystem, für den Sie Lese- und Schreibberechtigungen haben.

Note

IDT unterstützt nicht die Ausführung durch mehrere Benutzer von einem gemeinsam genutzten Speicherort aus, z. B. einem NFS-Verzeichnis oder einem freigegebenen Windows-Netzwerkordner. Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren und die IDT-Binärdatei auf Ihrer lokalen Workstation auszuführen. Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, extrahieren Sie IDT in ein Stammverzeichnis wie C:\ oder D:\, um Ihre Pfade unter der Grenze von 260 Zeichen zu halten.

Erstellen und konfigurieren Sie ein AWS-Konto

Bevor Sie IDT für verwenden können AWS IoT Greengrass, müssen Sie die folgenden Schritte ausführen:

1. [Erstellen Sie ein AWS-Konto](#). Wenn Sie bereits eine haben AWS-Konto, fahren Sie mit Schritt 2 fort.
2. [Konfigurieren Sie die Berechtigungen für IDT](#).

Mit diesen Kontoberechtigungen kann IDT in Ihrem Namen auf AWS Dienste zugreifen und AWS Ressourcen wie AWS IoT Dinge, Greengrass-Gruppen und Lambda-Funktionen erstellen.

Um diese Ressourcen zu erstellen, AWS IoT Greengrass verwendet IDT für die in der `config.json` Datei konfigurierten AWS Anmeldeinformationen, um API-Aufrufe in Ihrem Namen durchzuführen. Diese Ressourcen werden zu verschiedenen Zeiten während eines Tests bereitgestellt.

Note

Obwohl die meisten Tests für das [kostenlose Kontingent von Amazon Web Services](#) in Frage kommen, müssen Sie bei der Registrierung eine Kreditkarte angeben AWS-Konto. Weitere

Informationen finden Sie unter [Warum benötige ich eine Zahlungsmethode, wenn mein Konto vom kostenlosen Kontingent abgedeckt ist?](#).

Schritt 1: Erstellen Sie ein AWS-Konto

In diesem Schritt erstellen und konfigurieren Sie eine AWS-Konto. Wenn Sie bereits eine haben AWS-Konto, fahren Sie mit fort [the section called “Schritt 2: Konfigurieren von Berechtigungen für IDT”](#).

Melde dich für eine an AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie [https://portal.aws.amazon.com/billing/die Anmeldung](https://portal.aws.amazon.com/billing/die-Anmeldung).
2. Folgen Sie den Online-Anweisungen.

Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontos wird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um [Aufgaben auszuführen, die Root-Benutzerzugriff erfordern](#).

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Sie können Ihre aktuellen Kontoaktivitäten jederzeit einsehen und Ihr Konto verwalten, indem Sie zu <https://aws.amazon.com> gehen und Mein Konto auswählen.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich [AWS-Managementkonsole](#) als Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter [Anmelden als Root-Benutzer](#) im AWS-Anmeldung -Benutzerhandbuch zu.

2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

Anweisungen finden Sie unter [Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer \(Konsole\)](#) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

Anweisungen finden Sie unter [Aktivieren AWS IAM Identity Center](#) im AWS IAM Identity Center - Benutzerhandbuch.

2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter [Benutzerzugriff mit der Standardeinstellung konfigurieren](#).AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie [im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal](#).

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

Anweisungen hierzu finden Sie unter [Berechtigungssatz erstellen](#) im AWS IAM Identity Center - Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

Eine genaue Anleitung finden Sie unter [Gruppen hinzufügen](#) im AWS IAM Identity Center - Benutzerhandbuch.

Schritt 2: Konfigurieren von Berechtigungen für IDT

In diesem Schritt konfigurieren Sie die Berechtigungen, die IDT for AWS IoT Greengrass verwendet, um Tests durchzuführen und IDT-Nutzungsdaten zu sammeln. Sie können das AWS-Managementkonsole oder AWS Command Line Interface (AWS CLI) verwenden, um eine IAM-Richtlinie und einen Testbenutzer für IDT zu erstellen und dann Richtlinien an den Benutzer anzuhängen. Wenn Sie bereits einen Testbenutzer für IDT erstellt haben, fahren Sie mit [the section called "Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests"](#) oder [the section called "Optional: Konfigurieren des Docker-Containers"](#) fort.

- [So konfigurieren Sie Berechtigungen für IDT \(Konsole\)](#):
- [So konfigurieren Sie Berechtigungen für IDT \(AWS CLI\)](#):

So konfigurieren Sie Berechtigungen für IDT (Konsole)

Gehen Sie folgendermaßen vor, um mithilfe der Konsole Berechtigungen für IDT für AWS IoT Greengrass zu konfigurieren.

1. Melden Sie sich bei der [IAM-Konsole](#) an.
2. Erstellen Sie eine vom Kunden verwaltete Richtlinie, die Berechtigungen zum Erstellen von Rollen mit bestimmten Berechtigungen erteilt.
 - a. Wählen Sie im Navigationsbereich Richtlinien und dann Richtlinie erstellen.
 - b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie.

JSON

```
{  
  "Version": "2012-10-17",
```



```

    "Statement": [
      {
        "Sid": "ManageRolePoliciesForIDTGreengrass",
        "Effect": "Allow",
        "Action": [
          "iam:DetachRolePolicy",
          "iam:AttachRolePolicy"
        ],
        "Resource": [
          "arn:aws:iam::*:role/idt-*",
          "arn:aws:iam::*:role/GreengrassServiceRole"
        ],
        "Condition": {
          "ArnEquals": {
            "iam:PolicyARN": [
              "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
              "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
              "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
            ]
          }
        }
      },
      {
        "Sid": "ManageRolesForIDTGreengrass",
        "Effect": "Allow",
        "Action": [
          "iam:CreateRole",
          "iam>DeleteRole",
          "iam:PassRole",
          "iam:GetRole"
        ],
        "Resource": [
          "arn:aws:iam::123456789012:role/idt-*",
          "arn:aws:iam::123456789012:role/GreengrassServiceRole"
        ]
      }
    ]
  }
}

```

 Important

Die folgende Richtlinie erteilt die Berechtigung zum Erstellen und Verwalten von Rollen, die IDT für AWS IoT Greengrass benötigt. Dazu gehören Berechtigungen zum Anhängen der folgenden AWS verwalteten Richtlinien:

- [AWSGreengrassResourceAccessRolePolicy](#)
- [Greengrass OTAUpdate ArtifactAccess](#)
- [AWSLambdaBasicExecutionRole](#)

- c. Wählen Sie Next: Tags (Weiter: Tags) aus.
 - d. Klicken Sie auf Weiter: Prüfen.
 - e. Geben Sie unter Name **IDTGreengrassIAMPermissions** ein. Überprüfen Sie unter Summary (Zusammenfassung) die von Ihrer Richtlinie gewährten Berechtigungen.
 - f. Wählen Sie Richtlinie erstellen aus.
3. Erstellen Sie einen IAM-Benutzer und fügen Sie die von IDT erforderlichen Berechtigungen für hinzu. AWS IoT Greengrass
 - a. Erstellen Sie einen IAM-Benutzer. Folgen Sie den Schritten 1 bis 5 unter [Erstellen von IAM-Benutzern \(Konsole\)](#) im IAM-Benutzerhandbuch.
 - b. Hängen Sie die Berechtigungen an Ihren IAM-Benutzer an:
 - i. Wählen Sie auf der Seite Berechtigungen festlegen die Option Bestehende Richtlinien direkt anhängen aus.
 - ii. Suchen Sie nach der IDTGreengrassIAMPermissionsRichtlinie, die Sie im vorherigen Schritt erstellt haben. Markieren Sie das Kontrollkästchen.
 - iii. Suchen Sie nach der AWSIoTDeviceTesterForGreengrassFullAccessRichtlinie. Markieren Sie das Kontrollkästchen.

 Note

Die [AWSIoTDeviceTesterForGreengrassFullAccess](#) ist eine AWS verwaltete Richtlinie, die die Berechtigungen definiert, die IDT benötigt, um AWS Ressourcen zu erstellen und darauf zuzugreifen, die für Tests verwendet

werden. Weitere Informationen finden Sie unter [the section called “AWS verwaltete Richtlinie für IDT”](#).

- c. Wählen Sie Weiter: Tags aus.
 - d. Wählen Sie Next: Review (Weiter: Überprüfen), um eine Zusammenfassung Ihrer Auswahlmöglichkeiten anzuzeigen.
 - e. Wählen Sie Create user (Benutzer erstellen) aus.
 - f. Um die Zugriffsschlüssel des Benutzers (Zugriffsschlüssel IDs und geheime Zugriffsschlüssel) anzuzeigen, wählen Sie neben dem Passwort und dem Zugriffsschlüssel die Option Anzeigen aus. Zum Speichern der Zugriffsschlüssel wählen Sie Download .csv (CSV-Datei herunterladen) aus und speichern die Datei an einem sicheren Speicherort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.
4. Nächster Schritt: Konfigurieren Sie Ihr [physisches Gerät](#).

So konfigurieren Sie Berechtigungen für IDT (AWS CLI)

Gehen Sie wie folgt vor, AWS CLI um Berechtigungen für IDT für AWS IoT Greengrass zu konfigurieren. Wenn Sie bereits Berechtigungen in der Konsole konfiguriert haben, fahren Sie mit [the section called “Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests”](#) oder [the section called “Optional: Konfigurieren des Docker-Containers”](#) fort.

1. Installieren und konfigurieren Sie das auf Ihrem Computer, AWS CLI falls es noch nicht installiert ist. Folgen Sie den Schritten [unter Installation von AWS CLI](#) im AWS Command Line Interface Benutzerhandbuch.

 Note

Das AWS CLI ist ein Open-Source-Tool, mit dem Sie über Ihre AWS Befehlszeilen-Shell mit Diensten interagieren können.

2. Erstellen Sie eine vom Kunden verwaltete Richtlinie, die Berechtigungen zum Verwalten von IDT- und AWS IoT Greengrass -Rollen erteilt.

Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ManageRolePoliciesForIDTGreengrass",
      "Effect": "Allow",
      "Action": [
        "iam:DetachRolePolicy",
        "iam:AttachRolePolicy"
      ],
      "Resource": [
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:role/GreengrassServiceRole"
      ],
      "Condition": {
        "ArnEquals": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
            "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
            "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
          ]
        }
      }
    },
    {
      "Sid": "ManageRolesForIDTGreengrass",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam>DeleteRole",
        "iam:PassRole",
        "iam:GetRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/idt-*",
        "arn:aws:iam::123456789012:role/GreengrassServiceRole"
      ]
    }
  ]
}
```

```
}
]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --
policy-document '{\"Version\": \"2012-10-17\",      \"Statement\":
  [{\"Sid\": \"ManageRolePoliciesForIDTGreengrass\", \"Effect\": \"Allow
\", \"Action\": [\"iam:DetachRolePolicy\", \"iam:AttachRolePolicy\"],
  \"Resource\": [\"arn:aws:iam::*:role/idt-*\", \"arn:aws:iam::*:role/
GreengrassServiceRole\"], \"Condition\": {\"ArnEquals\": {\"iam:PolicyARN\":
  [\"arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
\", \"arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess
\", \"arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole\"]}}},
  {\"Sid\": \"ManageRolesForIDTGreengrass\", \"Effect\": \"Allow\", \"Action\":
  [\"iam:CreateRole\", \"iam>DeleteRole\", \"iam:PassRole\", \"iam:GetRole
\"], \"Resource\": [\"arn:aws:iam::*:role/idt-*\", \"arn:aws:iam::*:role/
GreengrassServiceRole\"]}}}'
```

Note

Dieser Schritt beinhaltet ein Beispiel für eine Windows-Eingabeaufforderung, da er eine andere JSON-Syntax als Linux-, macOS- oder Unix-Terminalbefehle verwendet.

3. Erstellen Sie einen IAM-Benutzer und fügen Sie die von IDT erforderlichen Berechtigungen für hinzu. AWS IoT Greengrass
 - a. Erstellen Sie einen IAM-Benutzer. In diesem Beispiel wird der Benutzer als IDTGreengrassUser bezeichnet.

```
aws iam create-user --user-name IDTGreengrassUser
```

- b. Hängen Sie die in Schritt 2 erstellte IDTGreengrassIAMPermissions Richtlinie an Ihren IAM-Benutzer an. Ersetzen Sie *<account-id>* den Befehl durch die ID Ihres AWS-Konto.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

- c. Hängen Sie die AWSIoTDeviceTesterForGreengrassFullAccess Richtlinie an Ihren IAM-Benutzer an.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
```

 Note

Dies [AWSIoTDeviceTesterForGreengrassFullAccess](#) ist eine AWS verwaltete Richtlinie, die die Berechtigungen definiert, die IDT benötigt, um AWS Ressourcen zu erstellen und auf sie zuzugreifen, die für Tests verwendet werden. Weitere Informationen finden Sie unter [the section called “AWS verwaltete Richtlinie für IDT”](#).

4. Erstellen Sie einen geheimen Zugriffsschlüssel für den Benutzer.

```
aws iam create-access-key --user-name IDTGreengrassUser
```

Speichern Sie die Ausgabe an einem sicheren Ort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.

5. Nächster Schritt: Konfigurieren Sie Ihr [physisches Gerät](#).

AWS verwaltete Richtlinie für AWS IoT Device Tester

Die [AWSIoTDeviceTesterForGreengrassFullAccess](#) verwaltete Richtlinie ermöglicht es IDT, Operationen auszuführen und Nutzungsmetriken zu sammeln. Diese Richtlinie gewährt die folgenden Berechtigungen:

- `iot-device-tester:CheckVersion`. Prüfen Sie AWS IoT Greengrass, ob eine Reihe von Testsuiten- und IDT-Versionen kompatibel sind.
- `iot-device-tester:DownloadTestSuite`. Laden Sie Testsuiten herunter.
- `iot-device-tester:LatestIdt`. Informieren Sie sich über die neueste IDT-Version, die zum Download zur Verfügung steht.
- `iot-device-tester:SendMetrics`. Veröffentlichen Sie Nutzungsdaten, die IDT über Ihre Tests sammelt.
- `iot-device-tester:SupportedVersion`. Rufen Sie die Liste der von IDT unterstützten Versionen der Suite ab AWS IoT Greengrass und testen Sie sie. Diese Informationen werden im Befehlszeilenfenster angezeigt.

Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests

Um Ihr Gerät zu konfigurieren, müssen Sie AWS IoT Greengrass Abhängigkeiten installieren, die AWS IoT Greengrass Core-Software konfigurieren, Ihren Host-Computer für den Zugriff auf Ihr Gerät konfigurieren und Benutzerberechtigungen auf Ihrem Gerät konfigurieren.

Überprüfen Sie die AWS IoT Greengrass Abhängigkeiten auf dem zu testenden Gerät

Bevor IDT for Ihre Geräte testen AWS IoT Greengrass kann, stellen Sie sicher, dass Sie Ihr Gerät wie unter [Erste Schritte mit AWS IoT Greengrass](#) beschrieben eingerichtet haben. Weitere Informationen zu unterstützten Plattformen finden Sie unter [Unterstützte Plattformen](#).

Konfigurieren Sie die Software AWS IoT Greengrass

IDT for AWS IoT Greengrass testet Ihr Gerät auf Kompatibilität mit einer bestimmten Version von AWS IoT Greengrass. IDT bietet zwei Optionen zum Testen AWS IoT Greengrass auf Ihren Geräten:

- Laden Sie eine Version der [AWS IoT Greengrass -Core-Software](#) herunter und verwenden Sie sie. IDT installiert die Software für Sie.
- Verwenden Sie eine Version der AWS IoT Greengrass Core-Software, die bereits auf Ihrem Gerät installiert ist.

Note

Jede Version von AWS IoT Greengrass hat eine entsprechende IDT-Version. Sie müssen die Version von IDT herunterladen, die der von AWS IoT Greengrass Ihnen verwendeten Version entspricht.

In den folgenden Abschnitten werden diese Optionen beschrieben. Sie müssen nur eine der Optionen ausführen.

Option 1: Laden Sie die AWS IoT Greengrass Core-Software herunter und konfigurieren Sie AWS IoT Device Tester für die Verwendung

Sie können die AWS IoT Greengrass Core-Software von der Download-Seite für die [AWS IoT Greengrass Kernsoftware](#) herunterladen.

1. Suchen Sie die richtige Architektur und Linux-Verteilung und wählen Sie dann Herunterladen aus.
2. Kopieren Sie die tar.gz-Datei in das Verzeichnis `<device-tester-extract-location>/products/greengrass/ggc`.

 Note

Ändern Sie nicht den Namen der Datei AWS IoT Greengrass tar.gz. Legen Sie nicht mehrere Dateien in diesem Verzeichnis für das gleiche Betriebssystem und die gleiche Architektur ab. Wenn Sie beispielsweise sowohl `greengrass-linux-armv7l-1.7.1.tar.gz`- als auch `greengrass-linux-armv7l-1.8.1.tar.gz`-Dateien in diesem Verzeichnis haben, werden die Tests fehlschlagen.

Option 2: Verwenden Sie eine bestehende Installation von AWS IoT Greengrass mit AWS IoT Device Tester

Konfigurieren Sie IDT so, dass die auf Ihrem Gerät installierte AWS IoT Greengrass Core-Software getestet wird, indem Sie das `greengrassLocation` Attribut zur `device.json` Datei im `<device-tester-extract-location>/configs` Ordner hinzufügen. Beispiel:

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

Weitere Informationen zur Datei `device.json` finden Sie unter [Konfigurieren von device.json](#).

Auf Linux-Geräten ist `/greengrass` der Standardspeicherort der AWS IoT Greengrass Core-Software.

 Note

Auf Ihrem Gerät sollte die AWS IoT Greengrass Core-Software installiert sein, die noch nicht gestartet wurde.

Stellen Sie sicher, dass Sie den Benutzer `ggc_user` und die `ggc_group` auf Ihrem Gerät hinzugefügt haben. Weitere Informationen finden Sie unter [Umgebungseinstellungen für AWS IoT Greengrass](#).

Konfigurieren des Host-Computers für den Zugriff auf das zu testende Gerät

IDT wird auf Ihrem Host-Computer ausgeführt und muss über SSH eine Verbindung mit Ihrem Gerät herstellen können. Es gibt zwei Möglichkeiten, IDT SSH-Zugriff auf Ihre zu testenden Geräte zu gewähren:

1. Befolgen Sie die Anweisungen hier, um ein SSH-Schlüsselpaar zu erstellen und Ihren Schlüssel zur Anmeldung bei Ihrem zu testenden Gerät ohne Angabe eines Passworts zu berechtigen.
2. Geben Sie einen Benutzernamen und ein Passwort für jedes Gerät in der Datei `device.json` an. Weitere Informationen finden Sie unter [Konfigurieren von device.json](#).

Sie können eine beliebige SSL-Implementierung verwenden, um einen SSH-Schlüssel zu erstellen. Die folgenden Anweisungen zeigen Ihnen, wie Sie [SSH-KEYGEN](#) oder [PuTTYgen](#) (für Windows) verwenden. Wenn Sie eine andere SSL-Implementierung verwenden, lesen Sie die Dokumentation zu dieser Implementierung.

IDT verwendet SSH-Schlüssel, um sich bei Ihrem zu testenden Gerät zu authentifizieren.

So erstellen Sie einen SSH-Schlüssel mit SSH-KEYGEN

1. Erstellen Sie einen SSH-Schlüssel.

Sie können mit dem Open SSH-Befehl `ssh-keygen` ein SSH-Schlüsselpaar erstellen. Wenn Sie bereits ein SSH-Schlüsselpaar auf Ihrem Host-Computer haben, ist es eine bewährte Methode, ein SSH-Schlüsselpaar speziell für IDT zu erstellen. Auf diese Weise kann Ihr Host-Computer nach Abschluss des Tests keine Verbindung mehr zu Ihrem Gerät herstellen, ohne ein Passwort einzugeben. Außerdem können Sie den Zugriff auf das Remote-Gerät auf die Personen beschränken, die tatsächlich Zugriff benötigen.

Note

Windows verfügt nicht über einen installierten SSH-Client. Weitere Informationen zur Installation eines SSH-Clients unter Windows finden Sie unter [Herunterladen der SSH-Client-Software](#).

Der Befehl `ssh-keygen` fordert Sie auf, einen Namen und Pfad zum Speichern des Schlüsselpaars einzugeben. Standardmäßig werden die Schlüsselpaardateien `id_rsa` (privater

Schlüssel) und `id_rsa.pub` (öffentlicher Schlüssel) genannt. Unter macOS und Linux ist der Standard-Speicherort dieser Dateien `~/.ssh/`. Unter Windows ist der Standard-Speicherort `C:\Users\<user-name>\.ssh`.

Wenn Sie dazu aufgefordert werden, geben Sie eine Schlüsselphrase zum Schutz Ihres SSH-Schlüssels ein. Weitere Informationen finden Sie unter [Generieren eines neuen SSH-Schlüssels](#).

2. Fügen Sie autorisierte SSH-Schlüssel zu Ihrem zu testenden Gerät hinzu.

IDT muss Ihren privaten SSH-Schlüssel für die Anmeldung bei Ihrem zu testenden Gerät verwenden. Um Ihren privaten SSH-Schlüssel für die Anmeldung bei Ihrem zu testenden Gerät zu autorisieren, verwenden Sie den Befehl `ssh-copy-id` von Ihrem Host-Computer. Dieser Befehl fügt Ihren öffentlichen Schlüssel zur Datei `~/.ssh/authorized_keys` auf dem zu testenden Gerät hinzu. Beispiel:

```
$ ssh-copy-id <remote-ssh-user>@<remote-device-ip>
```

Wo *remote-ssh-user* wird der Benutzername verwendet, um sich auf Ihrem zu testenden Gerät anzumelden, und *remote-device-ip* ist die IP-Adresse des zu testenden Geräts, mit dem Tests durchgeführt werden sollen. Beispiel:

```
ssh-copy-id pi@192.168.1.5
```

Wenn Sie dazu aufgefordert werden, geben Sie das Passwort für den im Befehl `ssh-copy-id` angegebenen Benutzernamen ein.

`ssh-copy-id` geht davon aus, dass der öffentliche Schlüssel `id_rsa.pub` heißt und am Standard-Speicherort gespeichert ist (`~/.ssh/` in macOS und Linux und `C:\Users\<user-name>\.ssh` in Windows). Wenn Sie dem öffentlichen Schlüssel einen anderen Namen gegeben oder ihn an einem anderen Ort gespeichert haben, müssen Sie den vollqualifizierten Pfad zu Ihrem öffentlichen SSH-Schlüssel mit der Option `-i` zu `ssh-copy-id` (z. B. `ssh-copy-id -i ~/my/path/myKey.pub`) angeben. Weitere Informationen zum Erstellen von SSH-Schlüsseln und Kopieren von öffentlichen Schlüsseln finden Sie unter [SSH-COPY-ID](#).

Um einen SSH-Schlüssel mit Pu zu erstellen TTYgen (nur Windows)

1. Stellen Sie sicher, dass der OpenSSH-Server und -Client auf Ihrem zu testenden Gerät installiert sind. Weitere Informationen finden Sie unter [OpenSSH](#).
2. Installieren Sie [PuTTYgen](#) auf Ihrem zu testenden Gerät.

3. Öffnen Sie PuTTYgen.
4. Wählen Sie Generate (Generieren) aus und bewegen Sie den Mauszeiger in das Feld, um einen privaten Schlüssel zu generieren.
5. Wählen Sie im Menü Conversions (Konvertierungen) die Option Export OpenSSH key (OpenSSH-Schlüssel exportieren) aus und speichern Sie den privaten Schlüssel mit der Dateierweiterung `.pem`.
6. Fügen Sie den öffentlichen Schlüssel der Datei `/home/<user>/.ssh/authorized_keys` auf dem zu testenden Gerät hinzu.
 - a. Kopieren Sie den Text des öffentlichen Schlüssels aus dem TTYgen Pu-Fenster.
 - b. Verwenden Sie PuTTY, um eine Sitzung auf Ihrem zu testenden Gerät zu erstellen.
 - i. Führen Sie in einer Eingabeaufforderung oder einem Windows Powershell-Fenster den folgenden Befehl aus:

`C:<path-to-putty>/putty.exe -ssh <user>@<dut-ip-address>`
 - ii. Wenn Sie dazu aufgefordert werden, geben Sie das Passwort Ihres Geräts ein.
 - iii. Verwenden Sie `vi` oder einen anderen Texteditor, um den öffentlichen Schlüssel an die Datei `/home/<user>/.ssh/authorized_keys` auf Ihrem zu testenden Gerät anzuhängen.
7. Aktualisieren Sie die Datei `device.json` mit Ihrem Benutzernamen, der IP-Adresse und dem Pfad zur Datei mit dem privaten Schlüssel, die Sie gerade auf Ihrem Host-Computer für jedes zu testende Gerät gespeichert haben. Weitere Informationen finden Sie unter [the section called "Konfigurieren von device.json"](#). Stellen Sie sicher, dass Sie den vollständigen Pfad und Dateinamen für den privaten Schlüssel angeben und Schrägstriche (`/`) verwenden. Verwenden Sie beispielsweise für den Windows-Pfad `C:\DT\privatekey.pem` die Angabe `C:/DT/privatekey.pem` in der Datei `device.json`.

Konfigurieren von Benutzerberechtigungen auf Ihrem Gerät

IDT führt Operationen für verschiedene Verzeichnisse und Dateien auf einem zu testenden Gerät aus. Einige dieser Operationen erfordern höhere Berechtigungen (mit `sudo`). Um diese Operationen zu automatisieren, AWS IoT Greengrass muss IDT for in der Lage sein, Befehle mit `sudo` auszuführen, ohne nach einem Passwort gefragt zu werden.

Führen Sie die folgenden Schritte auf dem zu testenden Gerät aus, um sudo den Zugriff ohne Aufforderung zur Eingabe eines Passworts zu erlauben.

 Note

username bezieht sich auf den SSH-Benutzer, der von IDT für den Zugriff auf das zu testende Gerät verwendet wird.

So fügen Sie den Benutzer der sudo-Gruppe hinzu

1. Führen Sie auf dem zu testenden Gerät `sudo usermod -aG sudo <username>` aus.
2. Melden Sie sich ab und melden Sie sich dann wieder an, damit die Änderungen wirksam werden.
3. Führen Sie `sudo echo test` aus, um zu überprüfen, ob der Benutzername erfolgreich hinzugefügt wurde. Wenn Sie nicht zur Eingabe eines Passworts aufgefordert werden, ist Ihr Benutzer korrekt konfiguriert.
4. Öffnen Sie die Datei `/etc/sudoers` und fügen Sie am Ende der Datei die folgende Zeile hinzu:

```
<ssh-username> ALL=(ALL) NOPASSWD: ALL
```

Konfigurieren Ihres Geräts zum Testen optionaler Funktionen

In den folgenden Themen wird beschrieben, wie Sie Ihre Geräte so konfigurieren, dass IDT-Tests für optionale Funktionen ausgeführt werden. Führen Sie diese Konfigurationsschritte nur aus, wenn Sie diese Funktionen testen möchten. Fahren Sie andernfalls mit dem Schritt [the section called “IDT-Einstellungen konfigurieren”](#) fort.

Themen

- [Optional: Konfiguration Ihres Docker-Containers für IDT für AWS IoT Greengrass](#)
- [Optional: Konfigurieren des Geräts für die ML-Qualifizierung](#)

Optional: Konfiguration Ihres Docker-Containers für IDT für AWS IoT Greengrass

AWS IoT Greengrass bietet ein Docker-Image und eine Docker-Datei, die es einfacher machen, die AWS IoT Greengrass Core-Software in einem Docker-Container auszuführen. Nachdem Sie den AWS IoT Greengrass Container eingerichtet haben, können Sie IDT-Tests ausführen. Derzeit werden nur x86_64 Docker-Architekturen unterstützt, um IDT für AWS IoT Greengrass auszuführen.

Für diese Funktion ist IDT v2.3.0 oder höher erforderlich.

Der Prozess der Einrichtung des Docker-Containers für die Ausführung von IDT-Tests hängt davon ab, ob Sie das von bereitgestellte Docker-Image oder die Docker-Datei verwenden. AWS IoT Greengrass

- [Verwenden des Docker-Images](#). Auf dem Docker-Image sind die Core-Software und die zugehörigen Abhängigkeiten installiert AWS IoT Greengrass .
- [Verwenden der Docker-Datei](#). Das Dockerfile enthält Quellcode, mit dem Sie benutzerdefinierte AWS IoT Greengrass Container-Images erstellen können. Das Abbild kann so geändert werden, dass es auf unterschiedlichen Plattformarchitekturen ausgeführt werden kann. Es kann auch verkleinert werden.

 Note

AWS IoT Greengrass stellt keine Dockerfiles oder Docker-Images für AWS IoT Greengrass die Kernsoftwareversion 1.11.1 bereit. Um IDT-Tests auf Ihren eigenen benutzerdefinierten Container-Images auszuführen, muss Ihr Image die Abhängigkeiten enthalten, die in der von bereitgestellten Dockerfile definiert sind. AWS IoT Greengrass

Die folgenden Funktionen sind nicht verfügbar, wenn Sie sie AWS IoT Greengrass in einem Docker-Container ausführen:

- [Konnektoren](#), die im Greengrass-Container-Modus ausgeführt werden. Um einen Konnektor in einem Docker-Container auszuführen, muss der Konnektor im Modus No container (Kein Container) ausgeführt werden. Informationen zum Suchen von Konnektoren, die den No Container (Kein Container)-Modus unterstützen, finden Sie unter [the section called “AWS-mitgelieferte Greengrass-Anschlüsse”](#). Einige dieser Konnektoren weisen einen Isolationsmodus-Parameter auf, den Sie auf No container (Kein Container) festlegen müssen.
- [Lokale Geräte- und Volume-Ressourcen](#) Ihre benutzerdefinierten Lambda-Funktionen, die im Docker-Container ausgeführt werden, müssen direkt auf Geräte und Volumes auf dem Core zugreifen.

Konfigurieren Sie das Docker-Image, bereitgestellt von AWS IoT Greengrass

Gehen Sie wie folgt vor, um das AWS IoT Greengrass Docker-Image für die Ausführung von IDT-Tests zu konfigurieren.

Voraussetzungen

Bevor Sie mit diesem Tutorial beginnen, müssen Sie Folgendes tun.

- Sie müssen die folgende Software und Versionen auf Ihrem Hostcomputer installieren, basierend auf der von Ihnen ausgewählten Version AWS Command Line Interface (AWS CLI).

AWS CLI version 2

- [Docker-Version](#) 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, wir empfehlen jedoch 18.09 oder höher.
- AWS CLI Version 2.0.0 oder höher.
 - Informationen zur Installation der AWS CLI Version 2 finden Sie unter [AWS CLI Version 2 installieren](#).
 - Informationen zur AWS CLI Konfiguration [von finden Sie unter Konfiguration von AWS CLI](#).

Note

Um auf einem Windows-Computer auf eine neuere AWS CLI Version 2 zu aktualisieren, müssen Sie den [MSI-Installationsvorgang](#) wiederholen.

AWS CLI version 1

- [Docker-Version](#) 18.09 oder höher. Frühere Versionen funktionieren möglicherweise auch, wir empfehlen jedoch 18.09 oder höher.
- [Python](#) Version 3.6 oder höher.
- [pip](#), Version 18.1 oder höher.
- AWS CLI Version 1.17.10 oder höher
 - Informationen zur Installation der AWS CLI Version 1 finden Sie unter [AWS CLI Version 1 installieren](#).
 - Informationen zur AWS CLI Konfiguration [von finden Sie unter Konfiguration von AWS CLI](#).
 - Führen Sie den folgenden Befehl aus, um auf die neueste AWS CLI Version von Version 1 zu aktualisieren.

```
pip install awscli --upgrade --user
```

 Note

Wenn Sie die [MSI-Installation](#) der AWS CLI Version 1 unter Windows verwenden, beachten Sie Folgendes:

- Wenn bei der Installation AWS CLI von Version 1 Botocore nicht installiert werden kann, versuchen Sie es mit der [Python- und Pip-Installation](#).
- Um auf eine neuere AWS CLI Version 1 zu aktualisieren, müssen Sie den MSI-Installationsvorgang wiederholen.

- Um auf Ressourcen von Amazon Elastic Container Registry (Amazon ECR) zugreifen zu können, müssen Sie die folgende Berechtigung erteilen.
- Amazon ECR verlangt von Benutzern, die `ecr:GetAuthorizationToken` Erlaubnis über eine AWS Identity and Access Management (IAM-) Richtlinie zu erteilen, bevor sie sich bei einer Registrierung authentifizieren und Bilder aus einem Amazon ECR-Repository per Push oder Pull abrufen können. Weitere Informationen finden Sie unter [Amazon ECR Repository-Richtlinienbeispiele](#) und [Zugriff auf ein Amazon ECR-Repository](#) im Amazon Elastic Container Registry-Benutzerhandbuch.

1. Laden Sie das Docker-Image herunter, und konfigurieren Sie den Container. Sie können das vorgefertigte Image von [Docker Hub](#) oder [Amazon Elastic Container Registry \(Amazon ECR\)](#) herunterladen und auf Windows-, macOS- und Linux-Plattformen (x86_64) ausführen.

Um das Docker-Image von Amazon ECR herunterzuladen, führen Sie alle Schritte unter aus. [the section called “Holen Sie sich das AWS IoT Greengrass Container-Image von Amazon ECR”](#) Kehren Sie dann zu diesem Thema zurück, um die Konfiguration fortzusetzen.

2. Nur für Linux-Benutzer: Stellen Sie sicher, dass der Benutzer, der IDT ausführt, über die Berechtigung zum Ausführen von Docker-Befehlen verfügt. Weitere Informationen finden Sie unter [Verwalten von Docker durch andere Benutzer als Stammbenutzer](#) in der Docker-Dokumentation.
3. Verwenden Sie den Befehl für Ihr Betriebssystem, um den AWS IoT Greengrass Container auszuführen:

Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
-v <host-path-to-kernel-config-file>:<container-path> \  
<image-repository>:<tag>
```

- *<host-path-to-kernel-config-file>* Ersetzen Sie durch den Pfad zur Kernel-Konfigurationsdatei auf dem Host und *<container-path>* durch den Pfad, unter dem das Volume im Container gemountet ist.

Die Kernel-Konfigurationsdatei auf dem Host befindet sich normalerweise in /proc/config.gz oder /boot/config-*<kernel-release-date>*. Sie können ausführen `uname -r`, um den *<kernel-release-date>* Wert zu finden.

Beispiel: Zum Mounten der Konfigurationsdatei von /boot/config-*<kernel-release-date>*

```
-v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
```

Beispiel: Zum Mounten der Konfigurationsdatei von /proc/config.gz

```
-v /proc/config.gz:/proc/config.gz \
```

- Ersetzen Sie *<image-repository>*: *<tag>* im Befehl durch den Namen des Repositorys und das Tag des Zielbildes.

Beispiel: Um auf die neueste Version der AWS IoT Greengrass Core-Software zu verweisen

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Führen Sie den folgenden Befehl aus, um die Liste der AWS IoT Greengrass Docker-Images abzurufen.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```


macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Ersetzen Sie *<image-repository>: <tag>* im Befehl durch den Namen des Repositorys und das Tag des Zielimages.

Beispiel: Um auf die neueste Version der AWS IoT Greengrass Core-Software zu verweisen

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Führen Sie den folgenden Befehl aus, um die Liste der AWS IoT Greengrass Docker-Images abzurufen:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Ersetzen Sie *<image-repository>: <tag>* im Befehl durch den Namen des Repositorys und das Tag des Zielimages.

Beispiel: Um auf die neueste Version der AWS IoT Greengrass Core-Software zu verweisen

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Führen Sie den folgenden Befehl aus, um die Liste der AWS IoT Greengrass Docker-Images abzurufen:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Important

Geben Sie beim Testen mit IDT nicht das `--entrypoint /greengrass-entrypoint.sh \` Argument an, das zum Ausführen des Images für den allgemeinen AWS IoT Greengrass Gebrauch verwendet wird.

4. Nächster Schritt: [Konfigurieren Sie Ihre AWS Anmeldeinformationen und device.json Datei.](#)

Konfigurieren Sie das Dockerfile, bereitgestellt von AWS IoT Greengrass

Gehen Sie wie folgt vor, um das aus dem AWS IoT Greengrass Dockerfile erstellte Docker-Image für die Ausführung von IDT-Tests zu konfigurieren.

1. Laden Sie das Dockerfile-Paket von [the section called “AWS IoT Greengrass Docker-Software”](#) auf Ihren Hostcomputer herunter und extrahieren Sie es.
2. Öffnen Sie README.md. Die nächsten drei Schritte beziehen sich auf Abschnitte in dieser Datei.
3. Stellen Sie sicher, dass Sie die Anforderungen im Abschnitt Voraussetzungen erfüllen.
4. Nur Linux-Benutzer: Führen Sie die Schritte Symlink- und Hardlink-Schutz aktivieren und Netzwerkweiterleitung aktivieren aus. IPv4
5. Um das Docker-Image zu erstellen, führen Sie alle Schritte in Schritt 1 aus. Erstellen Sie das AWS IoT Greengrass Docker-Image. Kehren Sie dann zu diesem Thema zurück, um die Konfiguration fortzusetzen.
6. Verwenden Sie den Befehl für Ihr Betriebssystem, um den AWS IoT Greengrass Container auszuführen:

Linux

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
-v <host-path-to-kernel-config-file>:<container-path> \  
<image-repository>:<tag>
```

- *<host-path-to-kernel-config-file>* Ersetzen Sie durch den Pfad zur Kernel-Konfigurationsdatei auf dem Host und *<container-path>* durch den Pfad, unter dem das Volume im Container gemountet ist.

Die Kernel-Konfigurationsdatei auf dem Host befindet sich normalerweise in /proc/config.gz oder /boot/config-*<kernel-release-date>*. Sie können ausführen `uname -r`, um den *<kernel-release-date>* Wert zu finden.

Beispiel: Zum Mounten der Konfigurationsdatei von /boot/config-*<kernel-release-date>*

```
-v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
```

Beispiel: Zum Mounten der Konfigurationsdatei von proc/config.gz

```
-v /proc/config.gz:/proc/config.gz \
```

- Ersetzen Sie *<image-repository>*: *<tag>* im Befehl durch den Namen des Repositorys und das Tag des Zielbildes.

Beispiel: Um auf die neueste Version der AWS IoT Greengrass Core-Software zu verweisen

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Führen Sie den folgenden Befehl aus, um die Liste der AWS IoT Greengrass Docker-Images abzurufen.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Ersetzen Sie `<image-repository>`: `<tag>` im Befehl durch den Namen des Repositorys und das Tag des Zielimages.

Beispiel: Um auf die neueste Version der AWS IoT Greengrass Core-Software zu verweisen

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Führen Sie den folgenden Befehl aus, um die Liste der AWS IoT Greengrass Docker-Images abzurufen:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \  
-p 8883:8883 \  
<image-repository>:<tag>
```

- Ersetzen Sie `<image-repository>`: `<tag>` im Befehl durch den Namen des Repositorys und das Tag des Zielimages.

Beispiel: Um auf die neueste Version der AWS IoT Greengrass Core-Software zu verweisen

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Führen Sie den folgenden Befehl aus, um die Liste der AWS IoT Greengrass Docker-Images abzurufen:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --  
repository-name aws-iot-greengrass
```

 Important

Geben Sie beim Testen mit IDT nicht das `--entrypoint /greengrass-entrypoint.sh` \ Argument an, das zum Ausführen des Images für den allgemeinen AWS IoT Greengrass Gebrauch verwendet wird.

7. Nächster Schritt: [Konfigurieren Sie Ihre AWS Anmeldeinformationen und device.json Datei.](#)

Fehlerbehebung bei Ihrem Docker-Container-Setup für IDT für AWS IoT Greengrass

Verwenden Sie die folgenden Informationen, um Probleme beim Ausführen eines Docker-Containers für IDT zu Testzwecken zu beheben. AWS IoT Greengrass

WARNUNG: Fehler beim Laden der Konfigurationfile: /home/user/.docker/config.json - stat /home/<user>/.docker/config.json: Berechtigung verweigert

Wenn Sie diesen Fehler beim Ausführen von `docker`-Befehlen unter Linux erhalten, führen Sie folgenden Befehl aus. Ersetzen Sie den folgenden Befehl durch `<user>` den Benutzer, der IDT ausführt.

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rxw /home/<user>/.docker -R
```

Optional: Konfigurieren des Geräts für die ML-Qualifizierung

IDT for AWS IoT Greengrass bietet Qualifizierungstests für maschinelles Lernen (ML) an, um zu überprüfen, ob Ihre Geräte ML-Inferenz lokal mithilfe von Cloud-trainierten Modellen durchführen können.

Um ML-Qualifizierungstests ausführen zu können, müssen Sie zunächst Ihre Geräte wie unter [the section called “Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests”](#) beschrieben konfigurieren. Führen Sie dann die Schritte in diesem Thema aus, um Abhängigkeiten für die ML-Frameworks zu installieren, die Sie ausführen möchten.

Zum Ausführen von Tests für die ML-Qualifizierung ist IDT v3.1.0 oder höher erforderlich.

Installieren von Abhängigkeiten für ML-Frameworks

Alle Abhängigkeiten für ML-Frameworks müssen im Verzeichnis `/usr/local/lib/python3.x/site-packages` installiert werden. Um sicherzustellen, dass sie im richtigen Verzeichnis installiert sind, empfehlen wir, bei der Installation der Abhängigkeiten sudo-Root-Berechtigungen zu verwenden. Virtuelle Umgebungen werden für Qualifizierungstests nicht unterstützt.

Note

Wenn Sie Lambda-Funktionen testen, die mit [Containerisierung \(im Greengrass-Containermodus\)](#) ausgeführt werden, wird das Erstellen von Symlinks für Python-Bibliotheken unter nicht unterstützt. `/usr/local/lib/python3.x` Um Fehler zu vermeiden, müssen Sie die Abhängigkeiten im richtigen Verzeichnis installieren.

Führen Sie die Schritte aus, um die Abhängigkeiten für Ihr Ziel-Framework zu installieren:

- [Installieren Sie MXNet Abhängigkeiten](#)
- [the section called “Installieren Sie TensorFlow Abhängigkeiten”](#)
- [Installieren von DLR-Abhängigkeiten](#)

Installieren MXNet Sie Apache-Abhängigkeiten

IDT-Qualifizierungstests für dieses Framework haben folgende Abhängigkeiten:

- Python 3.6 oder Python 3.7.

Note

Wenn Sie Python 3.6 verwenden, müssen Sie einen symbolischen Link von Python 3.7- zu Python 3.6 Binärdateien erstellen. Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt. Zum Beispiel:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- Apache MXNet v1.2.1 oder höher.

- NumPy. Die Version muss mit Ihrer MXNet Version kompatibel sein.

Wird installiert MXNet

Folgen Sie zur [Installation](#) den Anweisungen in der MXNet Dokumentation MXNet.

 Note

Wenn sowohl Python 2.x als auch Python 3.x auf Ihrem Gerät installiert ist, verwenden Sie Python 3.x in den Befehlen, die Sie zum Installieren der Abhängigkeiten ausführen.

Die Installation wird validiert MXNet

Wählen Sie eine der folgenden Optionen, um die MXNet Installation zu validieren.

Option 1: SSH in Ihr Gerät integrieren und Skripts ausführen

1. Integrieren Sie SSH in Ihr Gerät.
2. Führen Sie die folgenden Skripts aus, um zu überprüfen, ob die Abhängigkeiten korrekt installiert sind.

```
sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

Die Ausgabe gibt die Versionsnummer aus und das Skript sollte fehlerfrei beendet werden.

Option 2: IDT-Abhängigkeitstest ausführen

1. Stellen Sie sicher, dass `device.json` für die ML-Qualifizierung konfiguriert ist. Weitere Informationen finden Sie unter [the section called “Konfigurieren von device.json für die ML-Qualifizierung”](#).
2. Führen Sie den Abhängigkeitstest für das Framework aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id mxnet_dependency_check
```

In der Testzusammenfassung wird das Ergebnis PASSED für `mldependencies` angezeigt.

Installieren Sie TensorFlow Abhängigkeiten

IDT-Qualifizierungstests für dieses Framework haben folgende Abhängigkeiten:

- Python 3.6 oder Python 3.7.

 Note

Wenn Sie Python 3.6 verwenden, müssen Sie einen symbolischen Link von Python 3.7- zu Python 3.6 Binärdateien erstellen. Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt. Zum Beispiel:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- TensorFlow 1.x.

Wird installiert TensorFlow

Folgen Sie den Anweisungen in der TensorFlow Dokumentation, um TensorFlow 1.x [mit Pip](#) oder [aus dem Quellcode](#) zu installieren.

 Note

Wenn sowohl Python 2.x als auch Python 3.x auf Ihrem Gerät installiert ist, verwenden Sie Python 3.x in den Befehlen, die Sie zum Installieren der Abhängigkeiten ausführen.

Die Installation wird validiert TensorFlow

Wählen Sie eine der folgenden Optionen, um die TensorFlow Installation zu validieren.

Option 1: SSH in Ihr Gerät integrieren und ein Skript ausführen

1. Integrieren Sie SSH in Ihr Gerät.
2. Führen Sie das folgende Skript aus, um zu überprüfen, ob die Abhängigkeit korrekt installiert ist.


```
sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
```

Die Ausgabe gibt die Versionsnummer aus und das Skript sollte fehlerfrei beendet werden.

Option 2: IDT-Abhängigkeitstest ausführen

1. Stellen Sie sicher, dass `device.json` für die ML-Qualifizierung konfiguriert ist. Weitere Informationen finden Sie unter [the section called “Konfigurieren von device.json für die ML-Qualifizierung”](#).
2. Führen Sie den Abhängigkeitstest für das Framework aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id tensorflow_dependency_check
```

In der Testzusammenfassung wird das Ergebnis PASSED für `mldependencies` angezeigt.

Installieren Sie die Abhängigkeiten von Amazon SageMaker AI Neo Deep Learning Runtime (DLR)

IDT-Qualifizierungstests für dieses Framework haben folgende Abhängigkeiten:

- Python 3.6 oder Python 3.7.

Note

Wenn Sie Python 3.6 verwenden, müssen Sie einen symbolischen Link von Python 3.7- zu Python 3.6 Binärdateien erstellen. Dadurch wird Ihr Gerät so konfiguriert, dass es die Python-Anforderung für AWS IoT Greengrass erfüllt. Zum Beispiel:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- SageMaker AI Neo DLR.
- numpy.

Nachdem Sie die DLR-Testabhängigkeiten installiert haben, müssen Sie [das Modell kompilieren](#).

Installieren von DLR

Befolgen Sie die Anweisungen in der DLR-Dokumentation, um [die Neo DLR zu installieren](#).

Note

Wenn sowohl Python 2.x als auch Python 3.x auf Ihrem Gerät installiert ist, verwenden Sie Python 3.x in den Befehlen, die Sie zum Installieren der Abhängigkeiten ausführen.

Validieren der DLR-Installation

Wählen Sie eine der folgenden Optionen, um die DLR-Installation zu validieren.

Option 1: SSH in Ihr Gerät integrieren und Skripts ausführen

1. Integrieren Sie SSH in Ihr Gerät.
2. Führen Sie die folgenden Skripts aus, um zu überprüfen, ob die Abhängigkeiten korrekt installiert sind.

```
sudo python3.7 -c "import dlr; print(dlr.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

Die Ausgabe gibt die Versionsnummer aus und das Skript sollte fehlerfrei beendet werden.

Option 2: IDT-Abhängigkeitstest ausführen

1. Stellen Sie sicher, dass `device.json` für die ML-Qualifizierung konfiguriert ist. Weitere Informationen finden Sie unter [the section called “Konfigurieren von device.json für die ML-Qualifizierung”](#).
2. Führen Sie den Abhängigkeitstest für das Framework aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id mldependencies --test-id dlr_dependency_check
```

In der Testzusammenfassung wird das Ergebnis PASSED für `mldependencies` angezeigt.

Kompilieren des DLR-Modells

Sie müssen das DLR-Modell kompilieren, bevor Sie es für ML-Qualifizierungstests verwenden können. Wählen Sie dazu eine der folgenden Optionen:

Option 1: Verwenden Sie Amazon SageMaker AI, um das Modell zu kompilieren

Gehen Sie wie folgt vor, um mithilfe von SageMaker KI das von IDT bereitgestellte ML-Modell zu kompilieren. Dieses Modell ist mit Apache vortrainiert. MXNet

1. Stellen Sie sicher, dass Ihr Gerätetyp von SageMaker KI unterstützt wird. Weitere Informationen finden Sie in den [Optionen für Zielgeräte](#) in der Amazon SageMaker AI API-Referenz. Wenn Ihr Gerätetyp derzeit nicht von SageMaker AI unterstützt wird, folgen Sie den Schritten unter [the section called “Option 2: Kompilieren des DLR-Modells mithilfe von TVM”](#).

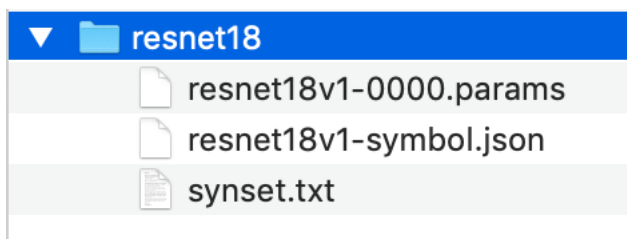
 Note

Die Ausführung des DLR-Tests mit einem von SageMaker KI kompilierten Modell kann 4 oder 5 Minuten dauern. Halten Sie IDT während dieser Zeit nicht an.

- Laden Sie die Tarball-Datei herunter, die das unkomplizierte, vortrainierte MXNet Modell für DLR enthält:

- [dlr-noncompiled-model-1.0.tar.gz](#)

3. Extrahieren Sie die Tarball-Datei. Dieser Befehl generiert die folgende Verzeichnisstruktur.



4. Verschieben Sie `synset.txt` aus dem Verzeichnis `resnet18` weg. Notieren Sie sich den neuen Speicherort. Sie kopieren diese Datei später in das kompilierte Modellverzeichnis.
5. Komprimieren Sie den Inhalt des Verzeichnisses `resnet18`.

```
tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
```

6. Laden Sie die komprimierte Datei in einen Amazon S3 S3-Bucket in Ihrem AWS-Konto hoch und folgen Sie dann den Schritten unter [Modell kompilieren \(Konsole\)](#), um einen Kompilierungsauftrag zu erstellen.
 - a. Verwenden Sie für Input configuration (Eingabekonfiguration) die folgenden Werte:
 - Geben Sie für Data input configuration (Dateneingabekonfiguration) den Wert `{"data": [1, 3, 224, 224]}` ein.
 - Wählen Sie für Machine Learning Framework (Machine-Learning-Framework) die Option MXNet.
 - b. Verwenden Sie für Output configuration (Ausgabekonfiguration) die folgenden Werte:
 - Geben Sie als S3-Ausgabespeicherort den Pfad zum Amazon S3 S3-Bucket oder -Ordner ein, in dem Sie das kompilierte Modell speichern möchten.
 - Wählen Sie für Target device (Zielgerät) Ihren Gerätetyp.
7. Laden Sie das kompilierte Modell vom angegebenen Ausgabespeicherort herunter und entpacken Sie die Datei.
8. Kopieren Sie `synset.txt` in das kompilierte Modellverzeichnis.
9. Ändern Sie den Namen des kompilierten Modellverzeichnisses in `resnet18`.

Das kompilierte Modellverzeichnis muss die folgende Verzeichnisstruktur aufweisen.



Option 2: Kompilieren des DLR-Modells mithilfe von TVM

Gehen Sie folgendermaßen vor, um das von IDT bereitgestellte ML-Modell mithilfe von TVM zu kompilieren. Dieses Modell ist mit Apache vortrainiert MXNet, sodass Sie es MXNet auf dem Computer oder Gerät installieren müssen, auf dem Sie das Modell kompilieren. Folgen Sie zur Installation MXNet den Anweisungen in der [MXNet Dokumentation](#).

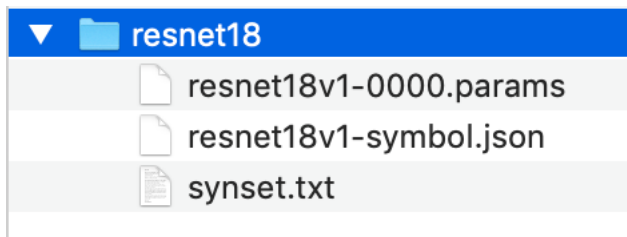
Note

Es wird empfohlen, das Modell auf dem Zielgerät zu kompilieren. Dies ist optional, kann jedoch zur Gewährleistung der Kompatibilität beitragen und potenzielle Probleme minimieren.

1. Laden Sie die Tarball-Datei herunter, die das unkompilierte, vortrainierte MXNet Modell für DLR enthält:

- [dlr-noncompiled-model-1.0.tar.gz](https://d1rnoncompiled-model-1.0.tar.gz)

2. Extrahieren Sie die Tarball-Datei. Dieser Befehl generiert die folgende Verzeichnisstruktur.



3. Befolgen Sie die Anweisungen in der TVM-Dokumentation, um [TVM aus der Quelle für Ihre Plattform zu erstellen und zu installieren](#).
4. Führen Sie nach der Erstellung von TVM die TVM-Kompilierung für das Modell resnet18 aus. Die folgenden Schritte basieren auf dem [Quick Start Tutorial for Compiling Deep Learning Models](#) in der TVM-Dokumentation.
 - a. Öffnen Sie die Datei `relay_quick_start.py` aus dem geklonten TVM-Repository.
 - b. Aktualisieren Sie den Code, der [ein neuronales Netzwerk im Relais definiert](#). Verwenden Sie eine der folgenden Optionen:
 - Option 1: Verwenden Sie `mxnet.gluon.model_zoo.vision.get_model`, um das Relaismodul und die Parameter zu erhalten:

```
from mxnet.gluon.model_zoo.vision import get_model
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

- Option 2: Kopieren Sie aus dem nicht kompilierten Modell, das Sie in Schritt 1 heruntergeladen haben, die folgenden Dateien in das gleiche Verzeichnis wie Datei `relay_quick_start.py`. Diese Dateien enthalten das Relaismodul und die Parameter.
 - `resnet18v1-symbol.json`
 - `resnet18v1-0000.params`
- c. Aktualisieren Sie den Code, der [das kompilierte Modul speichert und lädt](#), sodass der folgende Code verwendet wird.

```
from tvn.contrib import util
path_lib = "deploy_lib.so"
# Export the model library based on your device architecture
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

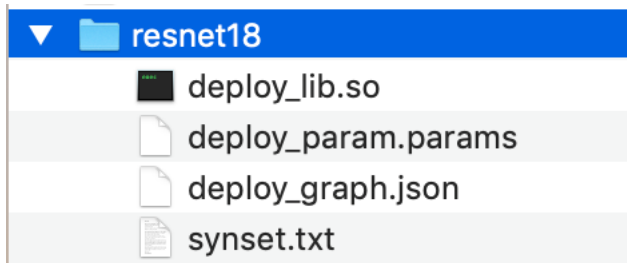
- d. Erstellen Sie das Modell:

```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

Dieser Befehl generiert die folgenden Dateien.

- `deploy_graph.json`
 - `deploy_lib.so`
 - `deploy_param.params`
5. Kopieren Sie die generierten Modelldateien in ein Verzeichnis mit dem Namen `resnet18`. Dies ist Ihr kompiliertes Modellverzeichnis.
 6. Kopieren Sie das kompilierte Modellverzeichnis auf Ihren Hostcomputer. Kopieren Sie dann `synset.txt` aus dem nicht kompilierten Modell, das Sie in Schritt 1 heruntergeladen haben, in das kompilierte Modellverzeichnis.

Das kompilierte Modellverzeichnis muss die folgende Verzeichnisstruktur aufweisen.



Als Nächstes [konfigurieren Sie Ihre Anmeldeinformationen und Ihre Datei](#). AWS device.json

Konfigurieren Sie die IDT-Einstellungen, um die AWS IoT Greengrass Qualification Suite auszuführen

Bevor Sie Tests ausführen, müssen Sie Einstellungen für AWS Anmeldeinformationen und Geräte auf Ihrem Hostcomputer konfigurieren.

Konfigurieren Sie Ihre AWS Anmeldeinformationen

Sie müssen Ihre IAM-Benutzeranmeldedaten in der *<device-tester-extract-location>* / configs/config.json Datei konfigurieren. Verwenden Sie die Anmeldeinformationen für den IDT für den AWS IoT Greengrass Benutzer, der in erstellt wurde. [the section called “Erstellen und konfigurieren Sie ein AWS-Konto”](#) Sie können Ihre Anmeldeinformationen auf zwei Arten angeben:

- Anmeldeinformationsdatei
- Umgebungsvariablen

Konfigurieren Sie die AWS Anmeldeinformationen mit einer Anmeldeinformationsdatei

IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter [Konfigurations- und Anmeldeinformationsdateien](#).

Der Speicherort der Datei mit den Anmeldeinformationen variiert je nach verwendetem Betriebssystem:

- macOS Linux: ~/.aws/credentials
- Windows: C:\Users*UserName*\.aws\credentials

Fügen Sie der credentials Datei Ihre AWS Anmeldeinformationen im folgenden Format hinzu:

```
[default]
aws_access_key_id = <your_access_key_id>
aws_secret_access_key = <your_secret_access_key>
```

Um IDT für die Verwendung von AWS Anmeldeinformationen aus Ihrer `credentials` Datei AWS IoT Greengrass zu konfigurieren, bearbeiten Sie Ihre `config.json` Datei wie folgt:

```
{
  "awsRegion": "us-west-2",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

Note

Wenn Sie das `default` AWS Profil nicht verwenden, ändern Sie unbedingt den Profilnamen in Ihrer `config.json` Datei. Weitere Informationen hierzu finden Sie unter [Benannte Profile](#).

Konfigurieren Sie AWS Anmeldeinformationen mit Umgebungsvariablen

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Sie werden nicht gespeichert, wenn Sie die SSH-Sitzung schließen. IDT for AWS IoT Greengrass kann die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und zum Speichern Ihrer AWS Anmeldeinformationen verwenden.

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie `export`:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit `set` festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```


Um den IDT so zu konfigurieren, dass er die Umgebungsvariablen verwendet, bearbeiten Sie den Abschnitt `auth` in Ihrer Datei `config.json`. Ein Beispiel:

```
{
  "awsRegion": "us-west-2",
  "auth": {
    "method": "environment"
  }
}
```

Konfigurieren von `device.json`

Zusätzlich zu den AWS Anmeldeinformationen AWS IoT Greengrass benötigt IDT für Informationen über die Geräte, auf denen Tests ausgeführt werden (z. B. IP-Adresse, Anmeldeinformationen, Betriebssystem und CPU-Architektur).

Sie müssen diese Informationen mittels der Vorlage `device.json` in `<device_tester_extract_location>/configs/device.json` angeben:

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      }
    ]
  }
]
```

```

    {
      "name": "streamManagement",
      "value": "yes | no"
    },
    {
      "name": "hsi",
      "value": "yes | no"
    },
    {
      "name": "ml",
      "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
    },
    ***** Remove the section below if the device is not qualifying for ML
    *****
    {
      "name": "mlLambdaContainerizationMode",
      "value": "container | process | both"
    },
    {
      "name": "processor",
      "value": "cpu | gpu"
    },
    ],
    ***** Remove the section below if the device is not qualifying for HSI
    *****
    "hsm": {
      "p11Provider": "/path/to/pkcs11ProviderLibrary",
      "slotLabel": "<slot_label>",
      "slotUserPin": "<slot_pin>",
      "privateKeyLabel": "<key_label>",
      "opensslEngine": "/path/to/openssl/engine"
    },
    ***** Remove the section below if the device is not qualifying for ML
    *****
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ]
    }
  }
}

```

```

    }
  ],
  "deviceResources": [
    {
      "name": "<resource-name>",
      "path": "<resource-path>",
      "type": "device | volume"
    }
  ]
},
*****
"kernelConfigLocation": "",
"greengrassLocation": "",
"devices": [
  {
    "id": "<device-id>",
    "connectivity": {
      "protocol": "ssh",
      "ip": "<ip-address>",
      "port": 22,
      "auth": {
        "method": "pki | password",
        "credentials": {
          "user": "<user-name>",
          "privKeyPath": "/path/to/private/key",
          "password": "<password>"
        }
      }
    }
  }
]
}
]

```

Note

Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.
Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
      },
      {
        "name": "docker",
        "value": "no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ***** Remove the section below if the device is not qualifying for ML
      *****
      {
        "name": "mlLambdaContainerizationMode",
        "value": "process"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      }
    ]
  }
]
```

```

    },

*****

    ],
    ***** Remove the section below if the device is not qualifying for ML
*****
    "machineLearning": {
        "dlrModelPath": "/path/to/compiled/dlr/model",
        "environmentVariables": [
            {
                "key": "<environment-variable-name>",
                "value": "<Path:$PATH>"
            }
        ],
        "deviceResources": [
            {
                "name": "<resource-name>",
                "path": "<resource-path>",
                "type": "device | volume"
            }
        ]
    },

*****

    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
        {
            "id": "<device-id>",
            "connectivity": {
                "protocol": "docker",
                "containerId": "<container-name | container-id>",
                "containerUser": "<user>"
            }
        }
    ]
}
]

```

Nachfolgend sind alle Pflichtfelder beschrieben:

id

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

sku

Ein alphanumerischer Wert, durch den das zu testende Gerät eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Boards nachzuverfolgen.

Note

Wenn Sie Ihr Motherboard im AWS Partner Gerätecatalog auflisten möchten, muss die hier angegebene SKU mit der SKU übereinstimmen, die Sie bei der Angebotserstellung verwenden.

features

Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Alle Funktionen sind erforderlich.

os und arch

Unterstützte Kombinationen aus Betriebssystem (OS) und Architektur:

- linux, x86_64
- linux, armv6l
- linux, armv7l
- linux, aarch64
- ubuntu, x86_64
- openwrt, armv7l
- openwrt, aarch64

 Note

Wenn Sie IDT verwenden, um die AWS IoT Greengrass Ausführung in einem Docker-Container zu testen, wird nur die x86_64-Docker-Architektur unterstützt.

container

Überprüft, ob das Gerät alle Software- und Hardwareanforderungen erfüllt, um Lambda-Funktionen im Containermodus auf einem Greengrass-Kern auszuführen.

Der gültige Wert ist oder. yes no

docker

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um den Greengrass Docker Application Deployment Connector für die Ausführung von Containern zu verwenden

Der gültige Wert ist oder. yes no

streamManagement

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um AWS IoT Greengrass Stream Manager auszuführen.

Der gültige Wert ist yes oderno.

hsi

Überprüft, ob die bereitgestellte gemeinsam genutzte HSI-Bibliothek eine Schnittstelle zum Hardware-Sicherheitsmodul (HSM) herstellen kann, und implementiert das erforderliche PKCS #11 korrekt. APIs Das HSM und die freigegebene Bibliothek müssen in der Lage sein, eine CSR zu signieren, TLS-Operationen auszuführen und die korrekten Schlüssellängen sowie den korrekten Algorithmus für den öffentlichen Schlüssel bereitzustellen.

Der gültige Wert ist oder. yes no

ml

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um die ML-Inferenz lokal ausführen zu können.

Der gültige Wert kann eine beliebige Kombination aus `mxnet`, `tensorflowdlr`, und sein `no` (z. B. `mxnet,mxnet,tensorflow,mxnet,tensorflow,dlr`, oder `no`).

`mlLambdaContainerizationMode`

Überprüft, ob das Gerät alle erforderlichen technischen Abhängigkeiten erfüllt, um ML-Inferenz im Container-Modus auf einem Greengrass-Gerät durchzuführen.

Der gültige Wert ist `container`, `process` oder `both`

`processor`

Überprüft, ob das Gerät alle Hardwareanforderungen für den angegebenen Prozessortyp erfüllt.

Der gültige Wert ist `cpu` oder `gpu`.

 Note

Wenn Sie die `ml` Funktion `container`, `docker`, oder nicht verwenden möchten `streamManagerhsi`, können Sie den entsprechenden Wert `value` auf festlegen `no`. Docker unterstützt nur die Funktionsqualifizierung für `streamManagement` und `ml`.

`machineLearning`

Optional. Konfigurationsinformationen für ML-Qualifizierungstests Weitere Informationen finden Sie unter [the section called “Konfigurieren von device.json für die ML-Qualifizierung”](#).

`hsm`

Optional. Konfigurationsinformationen für Tests mit einem AWS IoT Greengrass Hardware Security Module (HSM). Andernfalls sollte die `hsm`-Eigenschaft weggelassen werden. Weitere Informationen finden Sie unter [Integration von Hardware-Sicherheit](#).

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`hsm.p11Provider`

Der absolute Pfad zur libdl-ladbaren Bibliothek der PKCS#11-Implementierung.

`hsm.slotLabel`

Das Slot-Label zur Identifizierung des Hardwaremoduls

`hsm.slotUserPin`

Die Benutzer-PIN, die zur Authentifizierung des AWS IoT Greengrass Kerns gegenüber dem Modul verwendet wurde.

`hsm.privateKeyLabel`

Das Label zur Identifizierung des Schlüssels im Hardwaremodul.

`hsm.openSSLEngine`

Der absolute Pfad zur .so-Datei der OpenSSL-Engine, um die PKCS#11-Unterstützung unter OpenSSL zu aktivieren. Wird vom AWS IoT Greengrass OTA-Update-Agenten verwendet.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Derzeit sind die einzigen unterstützten Werte `ssh` für physische Geräte und `docker` für Docker-Container.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- pki
- password

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung beim zu testenden Gerät verwendet wird.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn sie auf gesetzt `connectivity.protocol` ist `ssh`.

`greengrassLocation`

Der Speicherort der AWS IoT Greengrass Core-Software auf Ihren Geräten.

Für physische Geräte wird dieser Wert nur verwendet, wenn Sie eine vorhandene Installation von verwenden AWS IoT Greengrass. Verwenden Sie dieses Attribut, um IDT anzuweisen, die auf Ihren Geräten installierte Version der AWS IoT Greengrass Core-Software zu verwenden.

Wenn Sie Tests in einem Docker-Container über das Docker-Image oder die von bereitgestellte Docker-Datei ausführen AWS IoT Greengrass, setzen Sie diesen Wert auf. `/greengrass kernelConfigLocation`

Optional. Der Pfad zur Kernel-Konfigurationsdatei. AWS IoT Device Tester verwendet diese Datei, um zu überprüfen, ob auf den Geräten die erforderlichen Kernel-Funktionen aktiviert sind. Falls nicht angegeben, verwendet IDT die folgenden Pfade, um nach der Kernel-Konfigurationsdatei zu suchen: `/proc/config.gz` und `/boot/config-<kernel-version>`. AWS IoT Device Tester verwendet den ersten Pfad, den er findet.

Konfigurieren von device.json für die ML-Qualifizierung

In diesem Abschnitt werden die optionalen Eigenschaften für die ML-Qualifizierung in der Gerätekonfigurationsdatei beschrieben. Wenn Sie Tests für die ML-Qualifizierung ausführen möchten, müssen Sie die Eigenschaften für Ihren Anwendungsfall definieren.

Sie können die Vorlage `device-ml.json` verwenden, um die Konfigurationseinstellungen für Ihr Gerät zu definieren. Diese Vorlage enthält die optionalen ML-Eigenschaften. Sie können auch die Datei `device.json` verwenden und die Eigenschaften für die ML-Qualifizierung hinzufügen. Diese Dateien sind unter `<device-tester-extract-location>/configs` gespeichert und enthalten Eigenschaften für die ML-Qualifizierung. Wenn Sie `device-ml.json` verwenden, müssen Sie die Datei in `device.json` umbenennen, bevor Sie IDT-Tests ausführen.

Weitere Informationen zu Eigenschaften für die Gerätekonfiguration, die nicht für die ML-Qualifizierung gelten, finden Sie unter [the section called “Konfigurieren von device.json”](#).

ml im Array features

Die ML-Frameworks, die Ihr Board unterstützt. Diese Eigenschaft erfordert IDT v3.1.0 oder höher.

- Wenn Ihr Board nur ein Framework unterstützt, geben Sie das Framework an. Beispiel:

```
{
  "name": "ml",
  "value": "mxnet"
}
```

- Wenn Ihr Board mehrere Frameworks unterstützt, geben Sie die Frameworks als durch Kommas getrennte Liste an. Beispiel:

```
{
  "name": "ml",
  "value": "mxnet,tensorflow"
}
```

mlLambdaContainerizationMode im Array features

Der [Containerisierungsmodus](#), den Sie für die Tests verwenden möchten. Diese Eigenschaft erfordert IDT v3.1.0 oder höher.

- Wählen Sie `process`, ob ML-Inferenzcode mit einer nicht containerisierten Lambda-Funktion ausgeführt werden soll. Für diese Option ist Version 1.10.x oder höher erforderlich. AWS IoT Greengrass
- Wählen Sie `container`, ob ML-Inferenzcode mit einer containerisierten Lambda-Funktion ausgeführt werden soll.
- Wählen Sie `both`, um ML-Inferenzcode mit beiden Modi auszuführen. Für diese Option ist Version 1.10.x oder höher erforderlich AWS IoT Greengrass .

processor im Array features

Gibt den Hardwarebeschleuniger an, den Ihr Board unterstützt. Diese Eigenschaft erfordert IDT v3.1.0 oder höher.

- Wählen Sie `cpu`, wenn Ihr Board eine CPU als Prozessor verwendet.
- Wählen Sie `gpu`, wenn Ihr Board eine GPU als Prozessor verwendet.

machineLearning

Optional. Konfigurationsinformationen für ML-Qualifizierungstests Diese Eigenschaft erfordert IDT v3.1.0 oder höher.

dllrModelPath

Erforderlich, um das Framework `dllr` verwenden zu können. Der absolute Pfad zu Ihrem kompilierten DLR-Modellverzeichnis, das mit `resnet18` benannt werden muss. Weitere Informationen finden Sie unter [the section called "Kompilieren des DLR-Modells"](#).

Note

Im Folgenden sehen Sie einen Beispielpfad unter macOS: `/Users/<user>/Downloads/resnet18`.

environmentVariables

Ein Array von Schlüssel-Wert-Paaren, die Einstellungen dynamisch an ML-Inferenztests übergeben können. Optional für CPU-Geräte. In diesem Abschnitt können Sie Framework-spezifische Umgebungsvariablen hinzufügen, die für Ihren Gerätetyp erforderlich sind. Weitere Informationen zu diesen Anforderungen finden Sie auf der offiziellen Website des Frameworks oder des Geräts. Um beispielsweise MXNet Inferenztests auf einigen Geräten auszuführen, sind möglicherweise die folgenden Umgebungsvariablen erforderlich.

```
"environmentVariables": [  
  ...  
  {  
    "key": "PYTHONPATH",  
    "value": "$MXNET_HOME/python:$PYTHONPATH"  
  },  
  {  
    "key": "MXNET_HOME",  
    "value": "$HOME/mxnet/"  
  },  
  ...  
]
```

Note

Das `value` Feld kann je nach Ihrer MXNet Installation variieren.

Wenn Sie Lambda-Funktionen testen, die mit [Containerisierung](#) auf GPU-Geräten ausgeführt werden, fügen Sie Umgebungsvariablen für die GPU-Bibliothek hinzu. Dadurch kann die GPU Berechnungen ausführen. Informationen zur Verwendung verschiedener GPU-Bibliotheken finden Sie in der offiziellen Dokumentation für die Bibliothek oder das Gerät.

Note

Konfigurieren Sie die folgenden Schlüssel, wenn für die Funktion `mlLambdaContainerizationMode` `container` oder `both` festgelegt ist

```
"environmentVariables": [  
  {
```

```

    "key": "PATH",
    "value": "<path/to/software/bin>:$PATH"
  },
  {
    "key": "LD_LIBRARY_PATH",
    "value": "<path/to/ld/lib>"
  },
  ...
]

```

deviceResources

Erforderlich für GPU-Geräte. Enthält [lokale Ressourcen](#), auf die über Lambda-Funktionen zugegriffen werden kann. Verwenden Sie diesen Abschnitt, um lokale Geräte- und Volume-Ressourcen hinzuzufügen.

- Geben Sie für Geräteressourcen "type": "device" an. Bei GPU-Geräten sollten die Geräteressourcen GPU-bezogene Gerätedateien unter /dev sein.

Note

Das Verzeichnis /dev/shm stellt eine Ausnahme dar. Sie kann nur als eine Volume-Ressource konfiguriert werden.

- Geben Sie für Volume-Ressourcen "type": "volume" an.

Führen Sie die AWS IoT Greengrass Qualifizierungssuite aus

Nachdem Sie [die gewünschte Konfiguration festgelegt haben](#), können Sie die Tests starten. Die Laufzeit der vollständigen Testsuite hängt von Ihrer Hardware ab. Zur Referenz: Es dauert etwa 30 Minuten, die vollständige Testsuite auf einem Raspberry Pi 3B auszuführen.

Die folgenden run-suite-Beispielbefehle zeigen, wie Sie die Qualifizierungstests für einen Gerätepool ausführen. Ein Gerätepool ist ein Satz identischer Geräte.

IDT v3.0.0 and later

Führen Sie alle Testgruppen in einer angegebenen Testsuite aus.

```

devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-id <pool-id>

```

Verwenden Sie den `list-suites`-Befehl, um die Testsuites aufzulisten, die sich im `tests`-Ordner befinden.

Führen Sie eine bestimmte Testgruppe in einer Testsuite aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-id <group-id> --pool-id <pool-id>
```

Verwenden Sie den `list-groups`-Befehl, um die Testgruppen in einer Testsuite aufzulisten.

Führen Sie einen bestimmten Testfall in einer Testgruppe aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-id <test-id>
```

Führen Sie mehrere Testfälle in einer Testgruppe aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --group-id <group-id> --test-id <test-id1>, <test-id2>
```

Listen Sie die Testfälle in einer Testgruppe auf.

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

Die Optionen für den `run-suite`-Befehl sind optional. Sie können z. B. `pool-id` weglassen, wenn in Ihrer `device.json`-Datei nur ein Gerätepool definiert ist. Sie können auch `suite-id` weglassen, wenn Sie die neueste Testsuite-Version im `tests`-Ordner ausführen möchten.

Note

IDT informiert Sie, wenn eine neuere Testsuite-Version online verfügbar ist. Weitere Informationen finden Sie unter [the section called “Legen Sie das standardmäßige Aktualisierungsverhalten fest”](#).

Weitere Informationen zu `run-suite` und anderen IDT-Befehlen finden Sie unter [the section called “IDT-Befehle”](#).

IDT v2.3.0 and earlier

Führen Sie alle Testgruppen in einer angegebenen Suite aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-id <pool-id>
```

Führen Sie eine bestimmte Testgruppe aus.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-id <group-id> --pool-id <pool-id>
```

`suite-id` und `pool-id` sind optional, wenn Sie eine einzelne Testsuite auf einem einzelnen Gerätepool ausführen. Dies bedeutet, dass Sie nur einen Gerätepool in Ihrer `device.json`-Datei definiert haben.

Suchen Sie nach Greengrass-Abhängigkeiten

Wir empfehlen, die Testgruppe für die Abhängigkeitsprüfung auszuführen, um sicherzustellen, dass alle Greengrass-Abhängigkeiten installiert sind, bevor Sie zugehörige Testgruppen ausführen. Zum Beispiel:

- Führen Sie vor Ausführung von Testgruppen für die Core-Qualifizierung `ggcdependencies` aus.
- Führen Sie `containerdependencies` vor dem Ausführen von containerspezifischen Testgruppen aus.
- Führen Sie `dockerdependencies` vor dem Ausführen von Docker-spezifischen Testgruppen aus.
- Führen Sie `ggcstreammanagementdependencies` vor für Stream Manager spezifischen Testgruppen aus.

Legen Sie das standardmäßige Aktualisierungsverhalten fest

Wenn Sie einen Testlauf starten, sucht IDT online nach einer neueren Testsuite-Version. Wenn eine solche verfügbar ist, fordert IDT Sie auf, auf die neueste verfügbare Version zu aktualisieren. Sie können das Flag `upgrade-test-suite` (oder `u`) setzen, um das Standardaktualisierungsverhalten zu steuern. Gültige Werte für sind:

- `y`. IDT lädt die neueste verfügbare Version herunter und verwendet sie.

- `n` (default). IDT verwendet die in der `suite-id`-Option angegebene Version. Wenn `suite-id` nicht angegeben, verwendet IDT die neueste Version im `tests` Ordner.

Wenn Sie das `upgrade-test-suite`-Flag nicht verwenden, informiert IDT Sie, wenn ein Update verfügbar ist, und wartet 30 Sekunden auf Ihre Eingabe (`y` oder `n`). Wenn keine Eingabe erfolgt, werden die Tests standardmäßig mit `n` fortgesetzt.

Die folgenden Beispiele zeigen häufig verwendete Anwendungsfälle für diese Funktion:

Automatisches Verwenden der neuesten Tests, die für eine Testgruppe verfügbar sind.

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

Ausführen von Tests in einer bestimmten Testsuite-Version.

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id DevicePool1
```

Aufforderung zur Aktualisierung zur Laufzeit.

```
devicetester_linux run-suite --pool-id DevicePool1
```

IDT für Befehle AWS IoT Greengrass

Die IDT-Befehle befinden sich im `<device-tester-extract-location>/bin`-Verzeichnis. Verwenden Sie sie für die folgenden Operationen:

IDT v3.0.0 and later

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Testsuite auf.

`list-suites`

Listet die verfügbaren Testsuites auf.

list-supported-products

Listet die unterstützten Produkte, in diesem Fall AWS IoT Greengrass Versionen, und Testsuite-Versionen für die aktuelle IDT-Version auf.

list-test-cases

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

run-suite

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden finden Sie einige unterstützte Optionen:

- `suite-id`. Die auszuführende Version der Testsuite. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als kommasetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.
- `test-id`. Die auszuführenden Testfälle als kommasetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`. Der zu testende Gerätepool. Sie müssen einen Pool angeben, wenn mehrere Gerätepools in der `device.json`-Datei definiert sind.
- `upgrade-test-suite`. Steuert, wie mit Versionsupdates der Testsuite umgegangen wird. Ab IDT v3.0.0 sucht IDT online nach aktualisierten Testsuite-Versionen. Weitere Informationen finden Sie unter [the section called "Test-Suite-Versionen"](#).
- `stop-on-first-failure`. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler gestoppt wird. Diese Option sollte mit `group-id` verwendet werden, um die angegebenen Testgruppen zu debuggen. Verwenden Sie diese Option nicht, wenn Sie eine vollständige Testsuite ausführen, um einen Qualifizierungsbericht zu generieren.
- `update-idt`. Legt die Antwort auf die Aufforderung zur Aktualisierung von IDT fest. Yals Eingabe stoppt die Testausführung, wenn IDT feststellt, dass es eine neuere Version gibt. Nals Eingabe wird die Testausführung fortgesetzt.
- `update-managed-policy`. Die Eingabe Y beendet die Testausführung, wenn IDT feststellt, dass die verwaltete Richtlinie des Benutzers nicht aktualisiert wird. Die Eingabe N setzt die Testausführung fort.

Weitere Informationen zu `run-suite`-Optionen erhalten Sie mit der `help`-Option:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

IDT v2.3.0 and earlier

help

Listet Informationen über den angegebenen Befehl auf.

list-groups

Listet die Gruppen in der jeweiligen Testsuite auf.

list-suites

Listet die verfügbaren Testsuites auf.

run-suite

Führt eine Reihe von Tests in einem Pool von Geräten aus.

Weitere Informationen zu run-suite-Optionen erhalten Sie mit der help-Option:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

Verstehen von Ergebnissen und Protokollen

In diesem Abschnitt wird beschrieben, wie Sie IDT-Ergebnisberichte und -Protokolle anzeigen und interpretieren können.

Anzeigen der Ergebnisse

Während der Ausführung schreibt IDT Fehler in die Konsole, Protokolldateien und Testberichte. Nachdem IDT die Qualifikations-Testsuite abgeschlossen hat, erstellt er zwei Testberichte. Diese Berichte befinden sich in *<device-tester-extract-location>/results/<execution-id>/*. Beide Berichte erfassen die Ergebnisse von der Ausführung der Qualifikations-Testsuite.

Dies `awsiotdevicetester_report.xml` ist der Qualifizierungstestbericht, den Sie einreichen AWS, um Ihr Gerät im AWS Partner Gerätecatalog aufzulisten. Die Bericht enthält die folgenden Elemente:

- Die IDT-Version.

- Die AWS IoT Greengrass Version, die getestet wurde.
- Die SKU- und der Gerätenamen, die in der `device.json`-Datei angegeben wurden.
- Die Funktionen des Gerätepools, der in der `device.json`-Datei angegeben wurde.
- Die aggregierte Zusammenfassung der Testergebnisse.
- Eine Aufschlüsselung der Testergebnisse nach Bibliotheken, die basierend auf den Geräteeigenschaften getestet wurden (z.B. lokaler Ressourcenzugriff, Shadow, MQTT etc.).

Der `GGQ_Result.xml` Bericht ist im [JUnit XML-Format](#). Sie können ihn in Continuous Integration and Deployment-Plattformen wie [Jenkins](#), [Bamboo](#) usw. integrieren. Der Bericht enthält die folgenden Elemente:

- Eine aggregierte Zusammenfassung der Testergebnisse.
- Aufschlüsselung der Testergebnisse nach den getesteten AWS IoT Greengrass Funktionen.

Interpretation von IDT-Berichten

Der Bericht im Abschnitt `awsiotdevicetester_report.xml` oder `awsiotdevicetester_report.xml` listet die Tests und die Ergebnisse auf, die ausgeführt wurden.

Im ersten XML-Tag `<testsuites>` ist die Zusammenfassung der Testausführung enthalten. Zum Beispiel:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0" disabled="0">
```

Im **`<testsuites>`**-Tag verwendete Attribute

name

Name der Testsuite

time

Zeit (in Sekunden), die zur Ausführung der Qualifikations-Suite erforderlich war

tests

Die Anzahl der ausgeführten Tests.

failures

Die Anzahl der ausgeführten Tests, die den Test nicht bestanden haben

errors

Die Anzahl der Tests, die IDT nicht ausführen konnte.

disabled

Dieses Attribut wird nicht verwendet und kann ignoriert werden.

Die Datei `awsiotdevicetester_report.xml` enthält ein `<awsproduct>`-Tag mit Informationen zum getesteten Produkt und den Produktfunktionen, die nach einer Reihe von Tests validiert wurden.

Im `<awsproduct>`-Tag verwendete Attribute

name

Der Name des getesteten Produkts.

version

Die Version des getesteten Produkts.

features

Die validierten Funktionen Als `required` gekennzeichnete Funktionen sind für die Einreichung Ihres Boards für die Qualifizierung erforderlich. Der folgende Ausschnitt zeigt, wie diese Informationen in der Datei `awsiotdevicetester_report.xml` angezeigt werden.

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></feature>
```

Als `optional` gekennzeichnete Funktionen sind für die Qualifizierung nicht erforderlich. Die folgenden Codeausschnitte zeigen optionale Funktionen.

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></feature>
```

Wenn keine Testfehler oder Fehler bei den erforderlichen Funktionen vorliegen, erfüllt Ihr Gerät die technischen Voraussetzungen für den Betrieb AWS IoT Greengrass und kann mit AWS IoT Diensten zusammenarbeiten. Wenn Sie Ihr Gerät im AWS Partner Gerätecatalog auflisten möchten, können Sie diesen Bericht als Qualifikationsnachweis verwenden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. Zum Beispiel:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>`-Tag, weist aber das Attribut `skipped` auf, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen `<testsuite>`-XML-Tags befinden sich `<testcase>`-Tags für alle ausgeführten Tests einer Testgruppe. Zum Beispiel:

```
<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>>
```

Im `<testcase>`-Tag verwendete Attribute

`name`

Der Name des Tests

`attempts`

Gibt an, wie oft IDT den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Zum Beispiel:

```
<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>
```

Anzeigen von -Protokollen

IDT generiert die Protokolle über die Testausführung in `<devicetester-extract-location>/results/<execution-id>/logs`. Es werden zwei Protokollgruppen generiert:

`test_manager.log`

Von der Test Manager-Komponente von AWS IoT Device Tester generierte Protokolle (z. B. Protokolle zur Konfiguration, Testsequenzierung und Berichtserstellung).

`<test_case_id>.log` (for example, `ota.log`)

Protokolle der Testgruppe, einschließlich Protokolle vom zu testenden Gerät. Wenn ein Test fehlschlägt, wird eine tar.gz-Datei mit den Protokollen des zu testenden Geräts für den Test erstellt (z. B. `ota_prod_test_1_ggc_logs.tar.gz`).

Weitere Informationen finden Sie unter [IDT zur Fehlerbehebung AWS IoT Greengrass](#).

Verwenden Sie IDT, um Ihre eigenen Testsuiten zu entwickeln und auszuführen

Ab IDT v4.0.0 AWS IoT Greengrass kombiniert IDT für ein standardisiertes Konfigurations-Setup und ein standardisiertes Ergebnisformat mit einer Testsuite-Umgebung, mit der Sie benutzerdefinierte Testsuiten für Ihre Geräte und Gerätesoftware entwickeln können. Sie können benutzerdefinierte Tests für Ihre eigene interne Validierung hinzufügen oder sie Ihren Kunden zur Geräteverifizierung zur Verfügung stellen.

Verwenden Sie IDT, um benutzerdefinierte Testsuiten wie folgt zu entwickeln und auszuführen:

Um benutzerdefinierte Testsuiten zu entwickeln

- Erstellen Sie Testsuiten mit benutzerdefinierter Testlogik für das Greengrass-Gerät, das Sie testen möchten.
- Stellen Sie IDT Ihre benutzerdefinierten Testsuiten für Testläufer zur Verfügung. Fügen Sie Informationen zu bestimmten Einstellungskonfigurationen für Ihre Testsuiten hinzu.

Um benutzerdefinierte Testsuiten auszuführen

- Richten Sie das Gerät ein, das Sie testen möchten.
- Implementieren Sie die Einstellungskonfigurationen entsprechend den Anforderungen der Testsuiten, die Sie verwenden möchten.

- Verwenden Sie IDT, um Ihre benutzerdefinierten Testsuiten auszuführen.
- Sehen Sie sich die Testergebnisse und Ausführungsprotokolle für die von IDT ausgeführten Tests an.

Laden Sie die neueste Version von AWS IoT Device Tester herunter für AWS IoT Greengrass

Laden Sie die [neueste Version](#) von IDT herunter und extrahieren Sie die Software an einen Speicherort in Ihrem Dateisystem, für den Sie Lese- und Schreibberechtigungen haben.

Note

IDT unterstützt nicht die Ausführung durch mehrere Benutzer von einem gemeinsam genutzten Speicherort aus, z. B. einem NFS-Verzeichnis oder einem freigegebenen Windows-Netzwerkordner. Es wird empfohlen, das IDT-Paket auf ein lokales Laufwerk zu extrahieren und die IDT-Binärdatei auf Ihrer lokalen Workstation auszuführen. Windows hat eine Pfadlängenbegrenzung von 260 Zeichen. Wenn Sie Windows verwenden, extrahieren Sie IDT in ein Stammverzeichnis wie C:\ oder D:\, um Ihre Pfade unter der Grenze von 260 Zeichen zu halten.

Workflow zur Erstellung einer Testsuite

Testsuiten bestehen aus drei Arten von Dateien:

- JSON-Konfigurationsdateien, die IDT Informationen zur Ausführung der Testsuite liefern.
- Testen Sie ausführbare Dateien, die IDT zur Ausführung von Testfällen verwendet.
- Zusätzliche Dateien, die zum Ausführen von Tests erforderlich sind.

Führen Sie die folgenden grundlegenden Schritte aus, um benutzerdefinierte IDT-Tests zu erstellen:

1. [Erstellen Sie JSON-Konfigurationsdateien](#) für Ihre Testsuite.
2. [Erstellen Sie ausführbare Testfalldateien](#), die die Testlogik für Ihre Testsuite enthalten.
3. Überprüfen und dokumentieren Sie die [Konfigurationsinformationen, die Testläufer benötigen](#), um die Testsuite auszuführen.

4. Stellen Sie sicher, dass IDT Ihre Testsuite ausführen und die [Testergebnisse](#) erwartungsgemäß liefern kann.

Folgen Sie den Anweisungen unter, um schnell eine benutzerdefinierte Beispielsuite zu erstellen und auszuführen. [Tutorial: IDT-Beispiel-Testsuite erstellen und ausführen](#)

Erste Schritte zum Erstellen einer benutzerdefinierten Testsuite in Python finden Sie unter [Tutorial: Entwickeln Sie eine einfache IDT-Testsuite](#).

Tutorial: IDT-Beispiel-Testsuite erstellen und ausführen

Der AWS IoT Device Tester-Download enthält den Quellcode für eine Beispiel-Testsuite. Sie können dieses Tutorial abschließen, um die Beispiel-Testsuite zu erstellen und auszuführen, damit Sie verstehen, wie Sie AWS IoT Device Tester für AWS IoT Greengrass die Ausführung benutzerdefinierter Testsuiten verwenden können.

In diesem Tutorial werden Sie die folgenden Schritte ausführen:

1. [Erstellen Sie die Beispiel-Testsuite](#)
2. [Verwenden Sie IDT, um die Beispieltestsuite auszuführen](#)

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Anforderungen an den Host-Computer
 - Aktuelle Version von AWS IoT Device Tester
 - [Python](#) 3.7 oder höher

Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

```
python3 --version
```

Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie `python --version` stattdessen. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um ihn `python3` als Alias für Ihren `python` Befehl festzulegen.

```
Set-Alias -Name "python3" -Value "python"
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter [Python herunterladen, um Python 3.7+ zu installieren](#). Weitere Informationen finden Sie in der [Python-Dokumentation](#).

- [urllib3](#)

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Installation korrekt urllib3 ist:

```
python3 -c 'import urllib3'
```

Wenn urllib3 es nicht installiert ist, führen Sie den folgenden Befehl aus, um es zu installieren:

```
python3 -m pip install urllib3
```

- Anforderungen an Speichergeräte
- Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Host-Computer.

Wir empfehlen Ihnen, einen [Raspberry Pi mit Raspberry](#) Pi OS zu verwenden. Stellen Sie sicher, dass Sie [SSH](#) auf Ihrem Raspberry Pi eingerichtet haben, um eine Remoteverbindung herzustellen.

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im `<device-tester-extract-location>/configs` Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "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>"
      }
    }
  }
}
```

Geben Sie im `devices` Objekt die folgenden Informationen ein:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

Note

Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.

Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Erstellen Sie die Beispiel-Testsuite

Der `<device-tester-extract-location>/samples/python` Ordner enthält Beispielfunktionsdateien, Quellcode und das IDT Client SDK, die Sie mithilfe der bereitgestellten Build-Skripten zu einer Testsuite kombinieren können. Die folgende Verzeichnisstruktur zeigt den Speicherort dieser Beispieldateien:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
```

```
#          ### build.sh
#          ### build.ps1
### sdks
### ...
### python
### idt_client
```

Um die Testsuite zu erstellen, führen Sie die folgenden Befehle auf Ihrem Host-Computer aus:

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
```

Dadurch wird die Beispiel-Testsuite in dem IDTSampleSuitePython_1.0.0 Ordner innerhalb des **<device-tester-extract-location>/tests** Ordners erstellt. Sehen Sie sich die Dateien im IDTSampleSuitePython_1.0.0 Ordner an, um zu verstehen, wie die Beispieltestsuite strukturiert ist, und sehen Sie sich verschiedene Beispiele für ausführbare Testfälle und JSON-Dateien für Testkonfigurationen an.

Nächster Schritt: Verwenden Sie IDT, um [die von Ihnen erstellte Beispieltestsuite auszuführen](#).

Verwenden Sie IDT, um die Beispieltestsuite auszuführen

Führen Sie die folgenden Befehle auf Ihrem Host-Computer aus, um die Beispiel-Testsuite auszuführen:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT führt die Beispieltestsuite aus und streamt die Ergebnisse an die Konsole. Wenn der Test abgeschlossen ist, sehen Sie die folgenden Informationen:

```
===== 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
```

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Abschließen des Tutorials zu lösen.

Der Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Problembehandlung des Testlaufs helfen kann. Stellen Sie sicher, dass Sie alle [Voraussetzungen](#) für dieses Tutorial erfüllen.

Es kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre `device.json` Datei enthält die richtige IP-Adresse, den richtigen Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Tutorial: Entwickeln Sie eine einfache IDT-Testsuite

Eine Testsuite kombiniert Folgendes:

- Testdateien, die die Testlogik enthalten
- JSON-Konfigurationsdateien, die die Testsuite beschreiben

Dieses Tutorial zeigt Ihnen, wie Sie IDT for verwenden AWS IoT Greengrass , um eine Python-Testsuite zu entwickeln, die einen einzigen Testfall enthält. In diesem Tutorial werden Sie die folgenden Schritte ausführen:

1. [Erstellen Sie ein Testsuite-Verzeichnis](#)
2. [Erstellen Sie JSON-Konfigurationsdateien](#)
3. [Erstellen Sie die ausführbare Testfalldatei](#)
4. [Führen Sie die Testsuite aus](#)

Voraussetzungen

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Anforderungen an den Host-Computer
 - Aktuelle Version von AWS IoT Device Tester
 - [Python](#) 3.7 oder höher

Führen Sie den folgenden Befehl aus, um die auf Ihrem Computer installierte Version von Python zu überprüfen:

```
python3 --version
```

Wenn die Verwendung dieses Befehls unter Windows einen Fehler zurückgibt, verwenden Sie `python --version` stattdessen. Wenn die zurückgegebene Versionsnummer 3.7 oder höher ist, führen Sie den folgenden Befehl in einem Powershell-Terminal aus, um ihn `python3` als Alias für Ihren `python` Befehl festzulegen.

```
Set-Alias -Name "python3" -Value "python"
```

Wenn keine Versionsinformationen zurückgegeben werden oder wenn die Versionsnummer kleiner als 3.7 ist, folgen Sie den Anweisungen unter [Python herunterladen, um Python 3.7+ zu installieren](#). Weitere Informationen finden Sie in der [Python-Dokumentation](#).

- [urllib3](#)

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob die Installation korrekt `urllib3` ist:

```
python3 -c 'import urllib3'
```

Wenn `urllib3` es nicht installiert ist, führen Sie den folgenden Befehl aus, um es zu installieren:

```
python3 -m pip install urllib3
```

- Anforderungen an Speichergeräte
- Ein Gerät mit einem Linux-Betriebssystem und einer Netzwerkverbindung zu demselben Netzwerk wie Ihr Host-Computer.

Wir empfehlen Ihnen, einen [Raspberry Pi mit Raspberry](#) Pi OS zu verwenden. Stellen Sie sicher, dass Sie [SSH](#) auf Ihrem Raspberry Pi eingerichtet haben, um eine Remoteverbindung herzustellen.

Erstellen Sie ein Testsuite-Verzeichnis

IDT unterteilt Testfälle innerhalb jeder Testsuite logisch in Testgruppen. Jeder Testfall muss sich innerhalb einer Testgruppe befinden. Erstellen Sie für dieses Tutorial einen Ordner mit dem Namen `MyTestSuite_1.0.0` und erstellen Sie in diesem Ordner den folgenden Verzeichnisbaum:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Erstellen Sie JSON-Konfigurationsdateien

Ihre Testsuite muss die folgenden erforderlichen [JSON-Konfigurationsdateien](#) enthalten:

Erforderliche JSON-Dateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe [Konfigurieren Sie suite.json](#).

`group.json`

Enthält Informationen über eine Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe [Konfigurieren Sie group.json](#).

test.json

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe [Konfigurieren Sie test.json](#).

1. Erstellen Sie in dem `MyTestSuite_1.0.0/suite` Ordner eine `suite.json` Datei mit der folgenden Struktur:

```
{
  "id": "MyTestSuite_1.0.0",
  "title": "My Test Suite",
  "details": "This is my test suite.",
  "userDataRequired": false
}
```

2. Erstellen Sie in dem `MyTestSuite_1.0.0/myTestGroup` Ordner eine `group.json` Datei mit der folgenden Struktur:

```
{
  "id": "MyTestGroup",
  "title": "My Test Group",
  "details": "This is my test group.",
  "optional": false
}
```

3. Erstellen Sie in dem `MyTestSuite_1.0.0/myTestGroup/myTestCase` Ordner eine `test.json` Datei mit der folgenden Struktur:

```
{
  "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"
        ]
    }
}
```

Der Verzeichnisbaum für Ihren MyTestSuite_1.0.0 Ordner sollte jetzt wie folgt aussehen:

```
MyTestSuite_1.0.0
### suite
### suite.json
### myTestGroup
### group.json
### myTestCase
### test.json
```

Holen Sie sich das IDT-Client-SDK

Sie verwenden das [IDT-Client-SDK](#), damit IDT mit dem zu testenden Gerät interagieren und Testergebnisse melden kann. Für dieses Tutorial verwenden Sie die Python-Version des SDK.

Kopieren Sie den *<device-tester-extract-location>*/sdks/python/ Ordner aus dem `idt_client` Ordner in Ihren `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner.

Führen Sie den folgenden Befehl aus, um zu überprüfen, ob das SDK erfolgreich kopiert wurde.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Erstellen Sie die ausführbare Testfalldatei

Die ausführbaren Testfalldateien enthalten die Testlogik, die Sie ausführen möchten. Eine Testsuite kann mehrere ausführbare Testfalldateien enthalten. Für dieses Tutorial erstellen Sie nur eine ausführbare Testfalldatei.

1. Erstellen Sie die Testsuite-Datei.

Erstellen Sie in dem `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Ordner eine `myTestCase.py` Datei mit dem folgenden Inhalt:

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

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

2. Verwenden Sie die Funktionen des Client-SDK, um Ihrer `myTestCase.py` Datei die folgende Testlogik hinzuzufügen:

a. Führen Sie auf dem zu testenden Gerät einen SSH-Befehl aus.

```
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. Senden Sie das Testergebnis an 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()
```

Geräteinformationen für IDT konfigurieren

Konfigurieren Sie Ihre Geräteinformationen für IDT, um den Test auszuführen. Sie müssen die `device.json` Vorlage im *<device-tester-extract-location>/configs* Ordner mit den folgenden Informationen aktualisieren.

```
[
  {
    "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>"
      }
    }
  }
]
]
```

Geben Sie im `devices` Objekt die folgenden Informationen ein:

`id`

Eine benutzerdefinierte eindeutige Kennung für Ihr Gerät.

`connectivity.ip`

Die IP-Adresse Ihres Geräts.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen zu Ihrem Gerät verwendet werden soll.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.user`

Der Benutzername, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zu dem privaten Schlüssel, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`devices.connectivity.auth.credentials.password`

Das Passwort, mit dem Sie sich auf Ihrem Gerät angemeldet haben.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

Note

Geben Sie `privKeyPath` nur an, wenn `method` auf `pki` gesetzt ist.

Geben Sie `password` nur an, wenn `method` auf `password` gesetzt ist.

Führen Sie die Testsuite aus

Nachdem Sie Ihre Testsuite erstellt haben, möchten Sie sicherstellen, dass sie wie erwartet funktioniert. Führen Sie dazu die folgenden Schritte aus, um die Testsuite mit Ihrem vorhandenen Gerätepool auszuführen.

1. Kopieren Sie Ihren `MyTestSuite_1.0.0` Ordner in `<device-tester-extract-location>/tests`.
2. Führen Sie die folgenden Befehle aus:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT führt Ihre Testsuite aus und streamt die Ergebnisse an die Konsole. Wenn der Test abgeschlossen ist, sehen Sie die folgenden Informationen:

```
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
```

Fehlerbehebung

Verwenden Sie die folgenden Informationen, um Probleme beim Abschließen des Tutorials zu lösen.

Der Testfall wird nicht erfolgreich ausgeführt

Wenn der Test nicht erfolgreich ausgeführt wird, streamt IDT die Fehlerprotokolle an die Konsole, die Ihnen bei der Problembehandlung des Testlaufs helfen kann. Bevor Sie die Fehlerprotokolle überprüfen, überprüfen Sie Folgendes:

- Das IDT-Client-SDK befindet sich im richtigen Ordner, wie in [diesem Schritt](#) beschrieben.
- Sie erfüllen alle [Voraussetzungen](#) für dieses Tutorial.

Es kann keine Verbindung zu dem zu testenden Gerät hergestellt werden

Überprüfen Sie Folgendes:

- Ihre `device.json` Datei enthält die richtige IP-Adresse, den richtigen Port und die richtigen Authentifizierungsinformationen.
- Sie können von Ihrem Host-Computer aus eine Verbindung zu Ihrem Gerät über SSH herstellen.

Erstellen Sie Konfigurationsdateien für die IDT-Testsuite

In diesem Abschnitt werden die Formate beschrieben, in denen Sie JSON-Konfigurationsdateien erstellen, die Sie beim Schreiben einer benutzerdefinierten Testsuite einbeziehen.

Erforderliche JSON-Dateien

`suite.json`

Enthält Informationen über die Testsuite. Siehe [Konfigurieren Sie suite.json](#).

`group.json`

Enthält Informationen über eine Testgruppe. Sie müssen für jede Testgruppe in Ihrer Testsuite eine `group.json` Datei erstellen. Siehe [Konfigurieren Sie group.json](#).

`test.json`

Enthält Informationen zu einem Testfall. Sie müssen für jeden Testfall in Ihrer Testsuite eine `test.json` Datei erstellen. Siehe [Konfigurieren Sie test.json](#).

Optionale JSON-Dateien

`state_machine.json`

Definiert, wie Tests ausgeführt werden, wenn IDT die Testsuite ausführt. Siehe [Konfigurieren Sie state_machine.json](#).

`userdata_schema.json`

Definiert das Schema für die [userdata.json](#) Datei, die Testläufer in ihre Einstellungskonfiguration aufnehmen können. Die `userdata.json` Datei wird für alle zusätzlichen Konfigurationsinformationen verwendet, die für die Ausführung des Tests erforderlich sind, aber nicht in der `device.json` Datei enthalten sind. Siehe [Konfigurieren Sie userdata_schema.json](#).

JSON-Konfigurationsdateien werden *<custom-test-suite-folder>* wie hier gezeigt in Ihrem abgelegt.

```
<custom-test-suite-folder>
### suite
  ### suite.json
  ### state_machine.json
  ### userdata_schema.json
  ### <test-group-folder>
    ### group.json
    ### <test-case-folder>
      ### test.json
```

Konfigurieren Sie suite.json

Die `suite.json` Datei legt Umgebungsvariablen fest und bestimmt, ob Benutzerdaten für die Ausführung der Testsuite erforderlich sind. Verwenden Sie die folgende Vorlage, um Ihre *<custom-test-suite-folder>/suite/suite.json* Datei zu konfigurieren:

```
{
  "id": "<suite-name>_<suite-version>",
  "title": "<suite-title>",
  "details": "<suite-details>",
  "userDataRequired": true | false,
  "environmentVariables": [
    {
      "key": "<name>",
      "value": "<value>",
    },
    ...
    {
      "key": "<name>",
      "value": "<value>",
    }
  ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

id

Eine eindeutige benutzerdefinierte ID für die Testsuite. Der Wert von `id` muss mit dem Namen des Testsuite-Ordners übereinstimmen, in dem sich die `suite.json` Datei befindet. Der Name der Suite und die Suite-Version müssen außerdem die folgenden Anforderungen erfüllen:

- `<suite-name>` darf keine Unterstriche enthalten.
- `<suite-version>` wird bezeichnet als `x.x.x`, wo `x` ist eine Zahl.

Die ID wird in IDT-generierten Testberichten angezeigt.

title

Ein benutzerdefinierter Name für das Produkt oder die Funktion, das von dieser Testsuite getestet wird. Der Name wird in der IDT-CLI für Testläufer angezeigt.

details

Eine kurze Beschreibung des Zwecks der Testsuite.

userDataRequired

Definiert, ob Testläufer benutzerdefinierte Informationen in eine `userdata.json` Datei aufnehmen müssen. Wenn Sie diesen Wert auf `true` setzen, müssen Sie die [userdata_schema.json](#) Datei auch in Ihren Testsuite-Ordner aufnehmen.

environmentVariables

Optional. Ein Array von Umgebungsvariablen, die für diese Testsuite festgelegt werden sollen.

environmentVariables.key

Der Name der Umgebungsvariable.

environmentVariables.value

Der Wert der Umgebungsvariable.

Konfigurieren Sie group.json

Die `group.json` Datei definiert, ob eine Testgruppe erforderlich oder optional ist. Verwenden Sie die folgende Vorlage, um Ihre `<custom-test-suite-folder>/suite/<test-group>/group.json` Datei zu konfigurieren:

```
{
```

```
"id": "<group-id>",
"title": "<group-title>",
"details": "<group-details>",
"optional": true | false,
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

id

Eine eindeutige benutzerdefinierte ID für die Testgruppe. Der Wert von `id` muss mit dem Namen des Testgruppenordners übereinstimmen, in dem sich die `group.json` Datei befindet, und darf keine Unterstriche () `_` enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

title

Ein beschreibender Name für die Testgruppe. Der Name wird in der IDT-CLI für Testläufer angezeigt.

details

Eine kurze Beschreibung des Zwecks der Testgruppe.

optional

Optional. Stellen Sie diese Option ein `true`, um diese Testgruppe als optionale Gruppe anzuzeigen, nachdem IDT die Ausführung der erforderlichen Tests abgeschlossen hat. Der Standardwert ist `false`.

Konfigurieren Sie test.json

Die `test.json` Datei bestimmt die ausführbaren Testfalldateien und die Umgebungsvariablen, die von einem Testfall verwendet werden. Weitere Hinweise zum Erstellen von ausführbaren Testfalldateien finden Sie unter [Ausführbare IDT-Testfalldateien erstellen](#)

Verwenden Sie die folgende Vorlage, um Ihre `<custom-test-suite-folder>/suite/<test-group>/<test-case>/test.json` Datei zu konfigurieren:

```
{
  "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>",
  }
]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`id`

Eine eindeutige benutzerdefinierte ID für den Testfall. Der Wert von `id` muss mit dem Namen des Testfallordners übereinstimmen, in dem sich die `test.json` Datei befindet, und darf keine Unterstriche (`_`) enthalten. Die ID wird in IDT-generierten Testberichten verwendet.

`title`

Ein beschreibender Name für den Testfall. Der Name wird in der IDT-CLI für Testläufer angezeigt.

`details`

Eine kurze Beschreibung des Zwecks des Testfalls.

`requireDUT`

Optional. `true` ist auf eingestellt, wenn ein Gerät für die Ausführung dieses Tests erforderlich ist, andernfalls auf `false`. Der Standardwert ist `true`. Testläufer konfigurieren die Geräte, mit denen sie den Test ausführen, in ihrer `device.json` Datei.

`requiredResources`

Optional. Ein Array, das Informationen über Ressourcengeräte bereitstellt, die für die Ausführung dieses Tests benötigt werden.

`requiredResources.name`

Der eindeutige Name, der dem Ressourcengerät gegeben werden soll, wenn dieser Test ausgeführt wird.

`requiredResources.features`

Eine Reihe von benutzerdefinierten Funktionen für Ressourcengeräte.

`requiredResources.features.name`

Der Name der Funktion. Die Gerätefunktion, für die Sie dieses Gerät verwenden möchten. Dieser Name wird mit dem Funktionsnamen abgeglichen, den der Testrunner in der `resource.json` Datei angegeben hat.

`requiredResources.features.version`

Optional. Die Version der Funktion. Dieser Wert wird mit der vom Test-Runner in der `resource.json` Datei bereitgestellten Feature-Version abgeglichen. Wenn keine Version bereitgestellt wird, wird die Funktion nicht überprüft. Wenn für die Funktion keine Versionsnummer erforderlich ist, lassen Sie dieses Feld leer.

`requiredResources.features.jobSlots`

Optional. Die Anzahl der gleichzeitigen Tests, die diese Funktion unterstützen kann. Der Standardwert ist 1. Wenn Sie möchten, dass IDT unterschiedliche Geräte für einzelne Funktionen verwendet, empfehlen wir Ihnen, diesen Wert auf 1 festzulegen.

`execution.timeout`

Die Zeit (in Millisekunden), die IDT wartet, bis der Test abgeschlossen ist. Weitere Informationen zum Einstellen dieses Werts finden Sie unter [Ausführbare IDT-Testfalldateien erstellen](#)

`execution.os`

Die ausführbaren Testfalldateien, die auf dem Betriebssystem des Host-Computers ausgeführt werden sollen, basieren auf dem Betriebssystem, auf dem IDT ausgeführt wird. Unterstützte Werte sind `linux`, `mac` und `win`.

`execution.os.cmd`

Der Pfad zu der ausführbaren Testfalldatei, die Sie für das angegebene Betriebssystem ausführen möchten. Dieser Speicherort muss sich im Systempfad befinden.

`execution.os.args`

Optional. Die Argumente, die zur Ausführung der ausführbaren Testfalldatei angegeben werden müssen.

`environmentVariables`

Optional. Ein Array von Umgebungsvariablen, die für diesen Testfall festgelegt wurden.

`environmentVariables.key`

Der Name der Umgebungsvariable.

`environmentVariables.value`

Der Wert der Umgebungsvariable.

 Note

Wenn Sie dieselbe Umgebungsvariable in der `test.json` Datei und in der `suite.json` Datei angeben, hat der Wert in der `test.json` Datei Vorrang.

Konfigurieren Sie state_machine.json

Eine Zustandsmaschine ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Sie bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge auf der Grundlage benutzerdefinierter Regeln und setzt den Übergang durch diese Zustände fort, bis der Endstatus erreicht ist.

Wenn Ihre Testsuite keine benutzerdefinierte Zustandsmaschine enthält, generiert IDT eine Zustandsmaschine für Sie. Die Standard-Zustandsmaschine erfüllt die folgenden Funktionen:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Weitere Informationen zur Funktionsweise der IDT-Zustandsmaschine finden Sie unter [Konfigurieren Sie die IDT-Zustandsmaschine](#).

Konfigurieren Sie userdata_schema.json

Die `userdata_schema.json` Datei bestimmt das Schema, in dem Testläufer Benutzerdaten bereitstellen. Benutzerdaten sind erforderlich, wenn Ihre Testsuite Informationen benötigt, die nicht in der `device.json` Datei enthalten sind. Beispielsweise benötigen Ihre Tests möglicherweise Anmeldeinformationen für Wi-Fi-Netzwerke, bestimmte offene Ports oder Zertifikate, die ein Benutzer bereitstellen muss. Diese Informationen können IDT als Eingabeparameter zur Verfügung gestellt werden `userdata`, dessen Wert eine `userdata.json` Datei ist, die Benutzer in ihrem `<device-tester-extract-location>/config` Ordner erstellen. Das Format der `userdata.json` Datei basiert auf der `userdata_schema.json` Datei, die Sie in die Testsuite aufnehmen.

Um anzugeben, dass Testläufer eine `userdata.json` Datei bereitstellen müssen:

1. Stellen Sie in der `suite.json` Datei `userDataRequired` auf ein `true`.
2. Erstellen Sie in Ihrem `<custom-test-suite-folder>` eine `userdata_schema.json` Datei.
3. Bearbeiten Sie die `userdata_schema.json` Datei, um ein gültiges [IETF-Draft v4-JSON-Schema](#) zu erstellen.

Wenn IDT Ihre Testsuite ausführt, liest es automatisch das Schema und verwendet es, um die vom Testläufer bereitgestellte `userdata.json` Datei zu validieren. Falls gültig, ist der Inhalt der `userdata.json` Datei sowohl im [IDT-Kontext als auch im State Machine-Kontext](#) verfügbar.

Konfigurieren Sie die IDT-Zustandsmaschine

Eine Zustandsmaschine ist ein Konstrukt, das den Ausführungsablauf der Testsuite steuert. Sie bestimmt den Startstatus einer Testsuite, verwaltet Zustandsübergänge auf der Grundlage benutzerdefinierter Regeln und setzt den Übergang durch diese Zustände fort, bis der Endstatus erreicht ist.

Wenn Ihre Testsuite keine benutzerdefinierte Zustandsmaschine enthält, generiert IDT eine Zustandsmaschine für Sie. Die Standard-Zustandsmaschine erfüllt die folgenden Funktionen:

- Bietet Testläufern die Möglichkeit, anstelle der gesamten Testsuite bestimmte Testgruppen auszuwählen und auszuführen.
- Wenn keine bestimmten Testgruppen ausgewählt sind, wird jede Testgruppe in der Testsuite in zufälliger Reihenfolge ausgeführt.
- Generiert Berichte und druckt eine Konsolenübersicht aus, in der die Testergebnisse für jede Testgruppe und jeden Testfall angezeigt werden.

Die Zustandsmaschine für eine IDT-Testsuite muss die folgenden Kriterien erfüllen:

- Jeder Status entspricht einer Aktion, die IDT ausführen muss, z. B. dem Ausführen einer Testgruppe oder eines Produkts oder einer Berichtsdatei.
- Beim Übergang zu einem Status wird die mit dem Status verknüpfte Aktion ausgeführt.
- Jeder Status definiert die Übergangsregel für den nächsten Status.
- Der Endstatus muss entweder `Succeed` oder `seinFail`.

Format der Zustandsmaschine

Sie können die folgende Vorlage verwenden, um Ihre eigene `<custom-test-suite-folder>/suite/state_machine.json` Datei zu konfigurieren:

```
{  
  "Comment": "<description>",
```



```
"StartAt": "<state-name>",
"States": {
  "<state-name>": {
    "Type": "<state-type>",
    // Additional state configuration
  }

  // Required states
  "Succeed": {
    "Type": "Succeed"
  },
  "Fail": {
    "Type": "Fail"
  }
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Comment

Eine Beschreibung der Zustandsmaschine.

StartAt

Der Name des Status, in dem IDT mit der Ausführung der Testsuite beginnt. Der Wert von StartAt muss auf einen der im States Objekt aufgelisteten Zustände gesetzt werden.

States

Ein Objekt, das benutzerdefinierte Statusnamen gültigen IDT-Staaten zuordnet. Jeder Bundesstaat. *state-name* Objekt enthält die Definition eines gültigen Zustands, der dem zugeordnet ist *state-name*.

Das States Objekt muss die Fail Zustände Succeed und enthalten. Hinweise zu gültigen Bundesstaaten finden Sie unter [Gültige Staaten und Bundesstaatendefinitionen](#).

Gültige Staaten und Bundesstaatendefinitionen

In diesem Abschnitt werden die Zustandsdefinitionen aller gültigen Staaten beschrieben, die in der IDT-Zustandsmaschine verwendet werden können. Einige der folgenden Staaten unterstützen Konfigurationen auf Testfallebene. Wir empfehlen jedoch, Regeln für den Statusübergang auf

Testgruppenebene statt auf Testfallebene zu konfigurieren, sofern dies nicht unbedingt erforderlich ist.

Definitionen von Bundesstaaten

- [RunTask](#)
- [Choice](#)
- [Parallel](#)
- [AddProductFeatures](#)
- [Bericht](#)
- [LogMessage](#)
- [SelectGroup](#)
- [Fehler](#)
- [Succeed](#)

RunTask

Der RunTask Staat führt Testfälle aus einer in der Testsuite definierten Testgruppe aus.

```
{
  "Type": "RunTask",
  "Next": "<state-name>",
  "TestGroup": "<group-id>",
  "TestCases": [
    "<test-id>"
  ],
  "ResultVar": "<result-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Next

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

TestGroup

Optional. Die ID der Testgruppe, die ausgeführt werden soll. Wenn dieser Wert nicht angegeben ist, führt IDT die Testgruppe aus, die der Testläufer auswählt.

TestCases

Optional. Ein Array von Testfällen IDs aus der in TestGroup angegebenen Gruppe.

Basierend auf den Werten von TestGroup und TestCases bestimmt IDT das Verhalten der Testausführung wie folgt:

- Wenn TestGroup sowohl als auch angegeben TestCases sind, führt IDT die angegebenen Testfälle aus der Testgruppe aus.
- Wenn TestCases angegeben, aber nicht angegeben TestGroup ist, führt IDT die angegebenen Testfälle aus.
- Wenn TestGroup angegeben, aber nicht angegeben TestCases ist, führt IDT alle Testfälle innerhalb der angegebenen Testgruppe aus.
- Wenn weder TestGroup oder angegeben TestCases ist, führt IDT alle Testfälle aus der Testgruppe aus, die der Testläufer aus der IDT-CLI auswählt. Um die Gruppenauswahl für Testläufer zu aktivieren, müssen Sie RunTask sowohl Status als auch Choice Status in Ihre statemachine.json Datei aufnehmen. Ein Beispiel dafür, wie das funktioniert, finden Sie unter [Beispiel für eine Zustandsmaschine: Vom Benutzer ausgewählte Testgruppen ausführen](#).

Weitere Informationen zur Aktivierung von IDT-CLI-Befehlen für Testläufer finden Sie unter [the section called “IDT-CLI-Befehle aktivieren”](#).

ResultVar

Der Name der Kontextvariablen, die mit den Ergebnissen des Testlaufs festgelegt werden soll. Geben Sie diesen Wert nicht an, wenn Sie keinen Wert für angegeben haben TestGroup. IDT legt den Wert der Variablen, die Sie definieren, auf true oder ResultVar auf der false Grundlage der folgenden Werte fest:

- Wenn der Variablenname die Form hat `text_text_passed`, wird der Wert darauf gesetzt, ob alle Tests in der ersten Testgruppe bestanden oder übersprungen wurden.
- In allen anderen Fällen wird der Wert darauf gesetzt, ob alle Tests in allen Testgruppen bestanden wurden oder ob sie übersprungen wurden.

In der Regel verwenden Sie RunTask state, um eine Testgruppen-ID anzugeben, ohne einen einzelnen Testfall anzugeben IDs, sodass IDT alle Testfälle in der angegebenen Testgruppe ausführt. Alle Testfälle, die von diesem Status ausgeführt werden, werden parallel in zufälliger Reihenfolge ausgeführt. Wenn jedoch für alle Testfälle ein Gerät ausgeführt werden muss und nur ein einziges Gerät verfügbar ist, werden die Testfälle stattdessen sequentiell ausgeführt.

Fehlerbehandlung

Wenn eine der angegebenen Testgruppen oder Testfälle nicht gültig ist, IDs gibt dieser Status den `RunTaskError` Ausführungsfehler aus. Wenn der Status auf einen Ausführungsfehler stößt, wird auch die `hasExecutionError` Variable im Zustandsmaschinenkontext auf `gesetzttrue`.

Choice

Mit `Choice` diesem Status können Sie basierend auf benutzerdefinierten Bedingungen dynamisch den nächsten Status festlegen, zu dem der Übergang erfolgen soll.

```
{
  "Type": "Choice",
  "Default": "<state-name>",
  "FallthroughOnError": true | false,
  "Choices": [
    {
      "Expression": "<expression>",
      "Next": "<state-name>"
    }
  ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Default

Der Standardstatus, in den der Übergang erfolgen soll, wenn keiner der in definierten Ausdrücke ausgewertet werden `Choices` kann. `true`

FallthroughOnError

Optional. Gibt das Verhalten an, wenn der Status bei der Auswertung von Ausdrücken auf einen Fehler stößt. Legt fest, `true` ob Sie einen Ausdruck überspringen möchten, wenn die Auswertung zu einem Fehler führt. Wenn keine Ausdrücke übereinstimmen, wechselt die Zustandsmaschine in den `Default` Status. Wenn der `FallthroughOnError` Wert nicht angegeben ist, wird standardmäßig der Wert verwendet. `false`

Choices

Eine Reihe von Ausdrücken und Zuständen, um zu bestimmen, in welchen Status nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Choices.Expression

Eine Ausdruckszeichenfolge, die einen booleschen Wert ergibt. Wenn der Ausdruck zu ausgewertet wird `true`, geht die Zustandsmaschine in den Zustand über, der in definiert ist. `Choices.Next` Ausdruckszeichenfolgen rufen Werte aus dem Zustandsmaschinen-Kontext ab und führen dann Operationen an ihnen durch, um einen booleschen Wert zu erhalten. Hinweise zum Zugriff auf den Zustandsmaschinenkontext finden Sie unter [Kontext der Zustandsmaschine](#)

Choices.Next

Der Name des Zustands, zu dem der Übergang erfolgen soll, wenn der in definierte Ausdruck zu `Choices.Expression` ausgewertet wird. `true`

Fehlerbehandlung

In den folgenden Fällen kann für den Choice Status eine Fehlerbehandlung erforderlich sein:

- Einige Variablen in den Auswahlausdrücken sind im Zustandsmaschinen-Kontext nicht vorhanden.
- Das Ergebnis eines Ausdrucks ist kein boolescher Wert.
- Das Ergebnis einer JSON-Suche ist keine Zeichenfolge, Zahl oder boolescher Wert.

In diesem Status können Sie keinen Catch Block verwenden, um Fehler zu behandeln. Wenn Sie die Ausführung der Zustandsmaschine beenden möchten, wenn sie auf einen Fehler stößt, müssen Sie `FallthroughOnError` auf `false` einstellen. Wir empfehlen jedoch, dass Sie die Einstellung `FallthroughOnError` auf `true` festlegen und je nach Anwendungsfall eine der folgenden Aktionen ausführen:

- Wenn davon ausgegangen wird, dass eine Variable, auf die Sie zugreifen, in einigen Fällen nicht existiert, verwenden Sie den Wert von `Default` und zusätzliche `Choices` Blöcke, um den nächsten Status anzugeben.
- Wenn eine Variable, auf die Sie zugreifen, immer existieren sollte, setzen Sie den `Default` Status auf `Fail`.

Parallel

Mit dem `Parallel` Status können Sie neue Zustandsmaschinen definieren und parallel zueinander ausführen.

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Branches": [
    <state-machine-definition>
  ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Next

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Branches

Eine Reihe von Zustandsmaschinendefinitionen, die ausgeführt werden sollen. Jede Zustandsmaschinen-Definition muss ihre eigenen `StartAtSucceed`, und `Fail` -Zustände enthalten. Die Zustandsmaschinendefinitionen in diesem Array können nicht auf Zustände verweisen, die außerhalb ihrer eigenen Definition liegen.

Note

Da jeder Zustandsmaschine denselben Zustandsmaschinenkontext verwendet, kann das Setzen von Variablen in einem Zweig und das anschließende Lesen dieser Variablen aus einem anderen Zweig zu unerwartetem Verhalten führen.

Der `Parallel` Status wechselt erst in den nächsten Status, nachdem er alle `Branch-State-Machines` ausgeführt hat. Jeder Status, für den ein Gerät erforderlich ist, wartet mit der Ausführung, bis das Gerät verfügbar ist. Wenn mehrere Geräte verfügbar sind, führt dieser Status Testfälle aus mehreren Gruppen parallel aus. Wenn nicht genügend Geräte verfügbar sind, werden die Testfälle sequentiell ausgeführt. Da Testfälle in zufälliger Reihenfolge ausgeführt werden, wenn sie parallel ausgeführt werden, können verschiedene Geräte verwendet werden, um Tests derselben Testgruppe auszuführen.

Fehlerbehandlung

Stellen Sie sicher, dass sowohl die Zweigzustandsmaschine als auch die übergeordnete Zustandsmaschine in den `Fail` Status wechseln, um Ausführungsfehler zu beheben.

Da Branch-State-Maschinen keine Ausführungsfehler an den übergeordneten Zustandsmaschinen übertragen, können Sie einen Catch Block nicht verwenden, um Ausführungsfehler in Branch-State-Machines zu behandeln. Verwenden Sie den `hasExecutionErrors` Wert stattdessen im Kontext des Shared State Machines. Ein Beispiel dafür, wie das funktioniert, finden Sie unter [Beispiel State Machine: Zwei Testgruppen parallel ausführen](#).

AddProductFeatures

Mit `AddProductFeatures` diesem Status können Sie der von IDT generierten `awsiotdevicetester_report.xml` Datei Produktmerkmale hinzufügen.

Bei einer Produktfunktion handelt es sich um benutzerdefinierte Informationen über bestimmte Kriterien, die ein Gerät möglicherweise erfüllt. Beispielsweise kann die MQTT Produktfunktion angeben, dass das Gerät MQTT-Nachrichten ordnungsgemäß veröffentlicht. Im Bericht werden Produktfunktionen als, oder als benutzerdefinierter Wert festgelegt `supported`/`not-supported`, je nachdem, ob die angegebenen Tests bestanden wurden.

Note

Der `AddProductFeatures` Staat generiert selbst keine Berichte. Dieser Status muss in den [ReportStatus](#) übergehen, in dem Berichte generiert werden können.

```
{
  "Type": "Parallel",
  "Next": "<state-name>",
  "Features": [
    {
      "Feature": "<feature-name>",
      "Groups": [
        "<group-id>"
      ],
      "OneOfGroups": [
        "<group-id>"
      ],
      "TestCases": [
        "<test-id>"
      ],
      "IsRequired": true | false,
      "ExecutionMethods": [
```

```

        "<execution-method>"
    ]
}
]
}

```

Nachfolgend sind alle Pflichtfelder beschrieben:

Next

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Features

Eine Reihe von Produktfunktionen, die in der `awsiotdevicetester_report.xml` Datei angezeigt werden sollen.

Feature

Der Name der Funktion

FeatureValue

Optional. Der benutzerdefinierte Wert, der anstelle von im Bericht verwendet werden soll `supported`. Wenn dieser Wert nicht angegeben ist, wird der Feature-Wert basierend auf den Testergebnissen auf `supported` oder `not-supported` gesetzt.

Wenn Sie einen benutzerdefinierten Wert für `FeatureValue` verwenden, können Sie dasselbe Feature mit unterschiedlichen Bedingungen testen, und IDT verkettet die Feature-Werte für die unterstützten Bedingungen. Der folgende Auszug zeigt beispielsweise das `MyFeature` Feature mit zwei separaten Feature-Werten:

```

...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},

```


...

Wenn beide Testgruppen erfolgreich sind, wird der Feature-Wert auf `first-feature-supported`, `second-feature-supported` gesetzt.

Groups

Optional. Ein Array von Testgruppen IDs. Alle Tests innerhalb jeder angegebenen Testgruppe müssen bestanden werden, damit die Funktion unterstützt wird.

OneOfGroups

Optional. Ein Array von Testgruppen IDs. Alle Tests innerhalb mindestens einer der angegebenen Testgruppen müssen bestanden werden, damit die Funktion unterstützt wird.

TestCases

Optional. Eine Reihe von Testfällen IDs. Wenn Sie diesen Wert angeben, gilt Folgendes:

- Alle angegebenen Testfälle müssen bestanden werden, damit die Funktion unterstützt wird.
- Groups darf nur eine Testgruppen-ID enthalten.
- OneOfGroups darf nicht angegeben werden.

IsRequired

Optional. Stellen Sie auf `false`, um diese Funktion im Bericht als optionale Funktion zu kennzeichnen. Der Standardwert ist `true`.

ExecutionMethods

Optional. Eine Reihe von Ausführungsmethoden, die dem in der `device.json` Datei angegebenen `protocol` Wert entsprechen. Wenn dieser Wert angegeben ist, müssen Testläufer einen `protocol` Wert angeben, der einem der Werte in diesem Array entspricht, um das Feature in den Bericht aufzunehmen. Wenn dieser Wert nicht angegeben wird, wird das Feature immer in den Bericht aufgenommen.

Um den `AddProductFeatures` Status verwenden zu können, müssen Sie den Wert von `ResultVar` in the `RunTask` state auf einen der folgenden Werte festlegen:

- Wenn Sie einen einzelnen Testfall angegeben haben IDs, setzen Sie ihn `ResultVar` auf `group-id_test-id_passed`.
- Wenn Sie keinen individuellen Testfall angegeben haben IDs, legen Sie `ResultVar` den Wert auf `festgroup-id_passed`.

Der AddProductFeatures Staat sucht auf folgende Weise nach Testergebnissen:

- Wenn Sie keinen Testfall angegeben haben IDs, wird das Ergebnis für jede Testgruppe anhand des Werts der `group-id_passed` Variablen im State-Machine-Kontext bestimmt.
- Wenn Sie einen Testfall angegeben haben IDs, wird das Ergebnis für jeden der Tests anhand des Werts der `group-id_test-id_passed` Variablen im Zustandsmaschinen-Kontext bestimmt.

Fehlerbehandlung

Wenn eine in diesem Status angegebene Gruppen-ID keine gültige Gruppen-ID ist, führt dieser Status zu einem AddProductFeaturesError Ausführungsfehler. Wenn im Status ein Ausführungsfehler auftritt, wird auch die `hasExecutionErrors` Variable im Zustandsmaschinenkontext auf `gesetzttrue`.

Bericht

Der Report Status generiert die `awsiotdevicetester_report.xml` Dateien `suite-name_Report.xml` und. In diesem Status wird der Bericht auch an die Konsole gestreamt.

```
{
  "Type": "Report",
  "Next": "<state-name>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Next

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Sie sollten immer gegen Ende der Testausführung in den Report Status wechseln, damit Testläufer die Testergebnisse einsehen können. In der Regel ist der nächste Status nach diesem StatusSucceed.

Fehlerbehandlung

Wenn in diesem Status Probleme beim Generieren der Berichte auftreten, wird der ReportError Ausführungsfehler ausgegeben.

LogMessage

Der LogMessage Status generiert die `test_manager.log` Datei und streamt die Protokollnachricht an die Konsole.

```
{
  "Type": "LogMessage",
  "Next": "<state-name>"
  "Level": "info | warn | error"
  "Message": "<message>"
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Next

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

Level

Die Fehlerstufe, auf der die Protokollnachricht erstellt werden soll. Wenn Sie eine ungültige Stufe angeben, generiert dieser Status eine Fehlermeldung und verwirft sie.

Message

Die zu protokollierende Nachricht.

SelectGroup

Der SelectGroup Status aktualisiert den Kontext der Zustandsmaschine, um anzugeben, welche Gruppen ausgewählt wurden. Die in diesem Status festgelegten Werte werden von allen nachfolgenden Choice Staaten verwendet.

```
{
  "Type": "SelectGroup",
  "Next": "<state-name>"
  "TestGroups": [
    <group-id>
  ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

Next

Der Name des Status, zu dem nach der Ausführung der Aktionen im aktuellen Status übergegangen werden soll.

TestGroups

Eine Reihe von Testgruppen, die als ausgewählt markiert werden. Für jede Testgruppen-ID in diesem Array wird die `group-id_selected` Variable `true` im Kontext auf gesetzt. Stellen Sie sicher, dass Sie eine gültige Testgruppe angeben IDs , da IDT nicht überprüft, ob die angegebenen Gruppen existieren.

Fehler

Der `Fail` Status gibt an, dass die Zustandsmaschine nicht korrekt ausgeführt wurde. Dies ist ein Endzustand für die Zustandsmaschine, und jede Zustandsmaschine muss diesen Zustand enthalten.

```
{
  "Type": "Fail"
}
```

Succeed

Der `Succeed` Status gibt an, dass die Zustandsmaschine korrekt ausgeführt wurde. Dies ist ein Endzustand für die Zustandsmaschine, und jede Zustandsmaschine muss diesen Zustand enthalten.

```
{
  "Type": "Succeed"
}
```

Kontext der Zustandsmaschine

Der Zustandsmaschinenkontext ist ein schreibgeschütztes JSON-Dokument, das Daten enthält, die der Zustandsmaschine während der Ausführung zur Verfügung stehen. Der Zustandsmaschinenkontext ist nur von der Zustandsmaschine aus zugänglich und enthält Informationen, die den Testablauf bestimmen. Sie können beispielsweise Informationen verwenden, die von Testläufern in der `userdata.json` Datei konfiguriert wurden, um zu ermitteln, ob ein bestimmter Test ausgeführt werden muss.

Der State-Machine-Kontext verwendet das folgende Format:

```
{
  "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

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Für einen ausgewählten Gerätepool werden diese Informationen aus dem entsprechenden Gerätepool-Array-Element der obersten Ebene abgerufen, das in der `device.json` Datei definiert ist.

userData

Informationen in der `userdata.json` Datei.

config

Informationen an die `config.json` Datei anheften.

suiteFailed

Der Wert wird auf den `false` Zeitpunkt gesetzt, zu dem die Zustandsmaschine gestartet wird. Wenn eine Testgruppe in einem `RunTask` Status ausfällt, wird dieser Wert `true` für die verbleibende Dauer der State-Machine-Ausführung auf gesetzt.

specificTestGroups

Wenn der Testläufer anstelle der gesamten Testsuite bestimmte Testgruppen zur Ausführung auswählt, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testgruppen IDs.

specificTestCases

Wenn der Testläufer anstelle der gesamten Testsuite bestimmte Testfälle zur Ausführung auswählt, wird dieser Schlüssel erstellt und enthält die Liste der spezifischen Testfälle IDs.

hasExecutionErrors

Wird nicht beendet, wenn die Zustandsmaschine gestartet wird. Wenn in einem Status ein Ausführungsfehler auftritt, wird diese Variable erstellt und `true` für die verbleibende Dauer der State-Machine-Ausführung auf „gesetzt“.

Sie können den Kontext mithilfe der JSONPath Notation abfragen. Die Syntax für JSONPath Abfragen in Statusdefinitionen lautet `{{$.query}}`. In einigen Bundesstaaten können Sie JSONPath Abfragen als Platzhalterzeichenfolgen verwenden. IDT ersetzt die Platzhalterzeichenfolgen durch den Wert der ausgewerteten JSONPath Abfrage aus dem Kontext. Sie können Platzhalter für die folgenden Werte verwenden:

- Der TestCases Wert in RunTask Bundesstaaten.
- Der Expression Choice Wertstatus.

Wenn Sie auf Daten aus dem State Machine-Kontext zugreifen, stellen Sie sicher, dass die folgenden Bedingungen erfüllt sind:

- Ihre JSON-Pfade müssen beginnen mit `$`.
- Jeder Wert muss eine Zeichenfolge, eine Zahl oder einen booleschen Wert ergeben.

Weitere Hinweise zur Verwendung der JSONPath Notation für den Zugriff auf Daten aus dem Kontext finden Sie unter: [Verwenden Sie den IDT-Kontext](#)

Ausführungsfehler

Ausführungsfehler sind Fehler in der Zustandsmaschinen-Definition, auf die der Zustandsmaschine bei der Ausführung eines Zustands stößt. IDT protokolliert Informationen zu jedem Fehler in der `test_manager.log` Datei und streamt die Protokollnachricht an die Konsole.

Sie können die folgenden Methoden verwenden, um Ausführungsfehler zu behandeln:

- Fügen Sie der Statusdefinition einen [CatchBlock](#) hinzu.
- Überprüfen Sie den [hasExecutionErrorsWert des Werts](#) im Kontext der Zustandsmaschine.

Fangen

Um es zu verwenden `Catch`, fügen Sie Ihrer Bundesstaatendefinition Folgendes hinzu:

```
"Catch": [
  {
    "ErrorEquals": [
      "<error-type>"
    ]
    "Next": "<state-name>"
  }
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

`Catch.ErrorEquals`

Eine Reihe von Fehlertypen, die abgefangen werden sollen. Wenn ein Ausführungsfehler mit einem der angegebenen Werte übereinstimmt, wechselt die Zustandsmaschine in den unter angegebenen `StatusCatch.Next`. In den einzelnen Statusdefinitionen finden Sie Informationen zur Art des Fehlers, den sie erzeugt.

`Catch.Next`

Der nächste Status, in den übergegangen werden soll, wenn im aktuellen Status ein Ausführungsfehler auftritt, der einem der in angegebenen Werte entspricht `Catch.ErrorEquals`.

`Catch`-Blöcke werden sequentiell behandelt, bis einer übereinstimmt. Wenn der Wert „Keine Fehler“ mit den in den `Catch`-Blöcken aufgelisteten Fehlern übereinstimmt, wird die Ausführung der Zustandsmaschinen fortgesetzt. Da Ausführungsfehler auf falsche Statusdefinitionen zurückzuführen sind, empfehlen wir, dass Sie in den Status `Fail` wechseln, wenn in einem Status ein Ausführungsfehler auftritt.

`hasExecutionError`

Wenn in einigen Staaten Ausführungsfehler auftreten, geben sie nicht nur den Fehler aus, sondern setzen den `hasExecutionError` Wert auch `true` im Kontext der Zustandsmaschine auf. Sie können diesen Wert verwenden, um zu erkennen, wann ein Fehler auftritt, und dann einen `Choice` Status verwenden, um die Zustandsmaschine in den `Fail` Status zu versetzen.

Diese Methode hat die folgenden Eigenschaften.

- Die Zustandsmaschine beginnt mit keinem Wert, der zugewiesen wurde `hasExecutionError`, und dieser Wert ist erst verfügbar, wenn ein bestimmter Status ihn festlegt. Das bedeutet, dass Sie den Status `false` für die Choice Staaten, die `FallthroughOnError` auf diesen Wert zugreifen, explizit auf `setzen` müssen, um zu verhindern, dass der Zustandsmaschine angehalten wird, wenn keine Ausführungsfehler auftreten.
- Sobald der Wert auf `gesetzt` ist `true`, `hasExecutionError` wird er niemals auf `False` gesetzt oder aus dem Kontext entfernt. Das bedeutet, dass dieser Wert nur nützlich ist, wenn er zum ersten Mal auf `gesetzt` wird `true`, und für alle nachfolgenden Zustände bietet er keinen aussagekräftigen Wert.
- Der `hasExecutionError` Wert wird von allen Zweigstatusmaschinen im `Parallel` Bundesstaat gemeinsam genutzt, was je nach Reihenfolge, in der auf ihn zugegriffen wird, zu unerwarteten Ergebnissen führen kann.

Aufgrund dieser Eigenschaften empfehlen wir nicht, diese Methode zu verwenden, wenn Sie stattdessen einen Catch-Block verwenden können.

Beispiel für Zustandsmaschinen

Dieser Abschnitt enthält einige Beispielkonfigurationen von Zustandsmaschinen.

Beispiele

- [Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe aus](#)
- [Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus](#)
- [Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus](#)
- [Beispiel State Machine: Zwei Testgruppen parallel ausführen](#)

Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe aus

Dieser Zustandsmaschine:

- Führt die Testgruppe mit der ID `ausGroupA`, die in der Suite in einer `group.json` Datei vorhanden sein muss.
- Prüft auf Ausführungsfehler und wechselt zu, `Fail` ob welche gefunden wurden.
- Generiert einen Bericht und wechselt zu, `Succeed` ob keine Fehler vorliegen, und `Fail` andernfalls.


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

Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus

Dieser Zustandsmaschine:

- Prüft, ob der Testläufer bestimmte Testgruppen ausgewählt hat. Die Zustandsmaschine sucht nicht nach bestimmten Testfällen, da Testläufer keine Testfälle auswählen können, ohne auch eine Testgruppe auszuwählen.
- Wenn Testgruppen ausgewählt sind:
 - Führt die Testfälle innerhalb der ausgewählten Testgruppen aus. Zu diesem Zweck spezifiziert die Zustandsmaschine nicht explizit Testgruppen oder Testfälle im RunTask Status.
 - Generiert einen Bericht, nachdem alle Tests ausgeführt und beendet wurden.
- Wenn keine Testgruppen ausgewählt sind:
 - Führt Tests in einer Testgruppe ausGroupA.
 - Generiert Berichte und beendet das Programm.

```
{
  "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"
    }
  }
}

```

Beispiel für eine Zustandsmaschine: Führen Sie eine einzelne Testgruppe mit Produktfunktionen aus

Dieser Zustandsmaschine:

- Führt die Testgruppe ausGroupA.
- Prüft auf Ausführungsfehler und wechselt, Fail ob welche gefunden wurden.
- Fügt der `awsiotdevicetester_report.xml` Datei das `FeatureThatDependsOnGroupA` Feature hinzu:

- Wenn der GroupA Wert erfolgreich ist, wird das Feature auf gesetztsupported.
- Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu, Succeed wenn keine Fehler vorliegen, und Fail andernfalls

```
{
  "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"
}
}
}

```

Beispiel State Machine: Zwei Testgruppen parallel ausführen

Diese Zustandsmaschine:

- Führt die Gruppen GroupA und die GroupB Testgruppen parallel aus. Die ResultVar Variablen, die im Kontext der RunTask Bundesstaaten gespeichert sind, stehen dem AddProductFeatures Staat zur Verfügung.
- Prüft auf Ausführungsfehler und wechselt, Fail falls welche gefunden wurden. Diese Zustandsmaschine verwendet keinen Catch Block, da diese Methode keine Ausführungsfehler in Zweigzustandsmaschinen erkennt.
- Fügt der awsiotdevicetester_report.xml Datei Funktionen hinzu, die auf den Gruppen basieren, die erfolgreich sind
 - Bei GroupA erfolgreicher Prüfung wird das Feature auf gesetztsupported.
 - Die Funktion ist im Bericht nicht als optional gekennzeichnet.
- Generiert einen Bericht und wechselt zu, Succeed wenn keine Fehler vorliegen, und Fail andernfalls

Wenn zwei Geräte im Gerätepool konfiguriert sind, GroupB können beide GroupA Geräte gleichzeitig ausgeführt werden. Wenn jedoch einer GroupA oder GroupB mehrere Tests enthalten sind, können beide Geräte diesen Tests zugewiesen werden. Wenn nur ein Gerät konfiguriert ist, werden die Testgruppen nacheinander ausgeführt.

```
{
```

```

"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"
  }
}
```

Ausführbare IDT-Testfalldateien erstellen

Sie können ausführbare Testfalldateien auf folgende Weise erstellen und in einem Testsuite-Ordner ablegen:

- Für Testsuiten, die Argumente oder Umgebungsvariablen aus den `test.json` Dateien verwenden, um zu bestimmen, welche Tests ausgeführt werden sollen, können Sie einen einzelnen ausführbaren Testfall für die gesamte Testsuite oder eine ausführbare Testdatei für jede Testgruppe in der Testsuite erstellen.
- Für eine Testsuite, in der Sie bestimmte Tests auf der Grundlage bestimmter Befehle ausführen möchten, erstellen Sie für jeden Testfall in der Testsuite eine ausführbare Testfalldatei.

Als Testautor können Sie bestimmen, welcher Ansatz für Ihren Anwendungsfall geeignet ist, und die ausführbare Testfalldatei entsprechend strukturieren. Stellen Sie sicher, dass Sie in jeder

`test.json` Datei den richtigen Pfad für die ausführbare Testfalldatei angeben und dass die angegebene ausführbare Datei korrekt ausgeführt wird.

Wenn alle Geräte für die Ausführung eines Testfalls bereit sind, liest IDT die folgenden Dateien:

- Der `test.json` für den ausgewählten Testfall festgelegte Prozess bestimmt, welche Prozesse gestartet und welche Umgebungsvariablen gesetzt werden sollen.
- Die `suite.json` für die Testsuite bestimmt die zu setzenden Umgebungsvariablen.

IDT startet den erforderlichen ausführbaren Testprozess auf der Grundlage der in der `test.json` Datei angegebenen Befehle und Argumente und übergibt die erforderlichen Umgebungsvariablen an den Prozess.

Verwenden Sie das IDT Client SDK

Mit dem IDT-Client SDKs können Sie das Schreiben von Testlogik in Ihre Testdatei mithilfe von API-Befehlen vereinfachen, die Sie verwenden können, um mit IDT und Ihren zu testenden Geräten zu interagieren. IDT bietet derzeit Folgendes: SDKs

- IDT-Client-SDK für Python
- IDT Client SDK for Go

Diese SDKs befinden sich im `<device-tester-extract-location>/sdks` Ordner. Wenn Sie eine neue ausführbare Testfalldatei erstellen, müssen Sie das SDK, das Sie verwenden möchten, in den Ordner kopieren, der Ihre ausführbare Testfalldatei enthält, und in Ihrem Code auf das SDK verweisen. Dieser Abschnitt enthält eine kurze Beschreibung der verfügbaren API-Befehle, die Sie in Ihren ausführbaren Testfalldateien verwenden können.

In diesem Abschnitt

- [Geräteinteraktion](#)
- [IDT-Interaktion](#)
- [Interaktion mit dem Host](#)

Geräteinteraktion

Mit den folgenden Befehlen können Sie mit dem zu testenden Gerät kommunizieren, ohne zusätzliche Funktionen für die Geräteinteraktion und das Konnektivitätsmanagement implementieren zu müssen.

ExecuteOnDevice

Ermöglicht es Testsuiten, Shell-Befehle auf einem Gerät auszuführen, das SSH- oder Docker-Shell-Verbindungen unterstützt.

CopyToDevice

Ermöglicht Testsuiten, eine lokale Datei vom Host-Computer, auf dem IDT ausgeführt wird, an einen bestimmten Speicherort auf einem Gerät zu kopieren, das SSH- oder Docker-Shell-Verbindungen unterstützt.

ReadFromDevice

Ermöglicht es Testsuiten, von der seriellen Schnittstelle von Geräten zu lesen, die UART-Verbindungen unterstützen.

Note

Da IDT keine direkten Verbindungen zu Geräten verwaltet, die mithilfe von Gerätezugriffsinformationen aus dem Kontext hergestellt werden, empfehlen wir, diese API-Befehle für Geräteinteraktionen in Ihren ausführbaren Testfalldateien zu verwenden. Wenn diese Befehle jedoch nicht Ihren Testfallanforderungen entsprechen, können Sie Gerätezugriffsinformationen aus dem IDT-Kontext abrufen und sie verwenden, um über die Testsuite eine direkte Verbindung zum Gerät herzustellen.

Um eine direkte Verbindung herzustellen, rufen Sie die Informationen in den `device.connectivity resource.devices.connectivity` Feldern für Ihr zu testendes Gerät bzw. für Ressourcengeräte ab. Weitere Informationen zur Verwendung des IDT-Kontextes finden Sie unter [Verwenden Sie den IDT-Kontext](#).

IDT-Interaktion

Die folgenden Befehle ermöglichen es Ihren Testsuiten, mit IDT zu kommunizieren.

PollForNotifications

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

GetContextValue und GetContextString

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter [Verwenden Sie den IDT-Kontext](#).

SendResult

Ermöglicht es Testsuiten, Testfallergebnisse an IDT zu melden. Dieser Befehl muss am Ende jedes Testfalls in einer Testsuite aufgerufen werden.

Interaktion mit dem Host

Mit dem folgenden Befehl können Ihre Testsuiten mit dem Host-Computer kommunizieren.

PollForNotifications

Ermöglicht Testsuiten, nach Benachrichtigungen von IDT zu suchen.

GetContextValue und GetContextString

Ermöglicht Testsuiten das Abrufen von Werten aus dem IDT-Kontext. Weitere Informationen finden Sie unter [Verwenden Sie den IDT-Kontext](#).

ExecuteOnHost

Ermöglicht Testsuiten die Ausführung von Befehlen auf dem lokalen Computer und ermöglicht IDT die Verwaltung des Lebenszyklus der ausführbaren Testfälle.

IDT-CLI-Befehle aktivieren

Der `run-suite` Befehl IDT CLI bietet mehrere Optionen, mit denen Test Runner die Testausführung anpassen kann. Damit Testläufer diese Optionen zum Ausführen Ihrer benutzerdefinierten Testsuite verwenden können, implementieren Sie Unterstützung für die IDT-CLI. Wenn Sie keine Unterstützung implementieren, können Testläufer weiterhin Tests ausführen, aber einige CLI-Optionen funktionieren nicht richtig. Um ein optimales Kundenerlebnis zu bieten, empfehlen wir, die Unterstützung für die folgenden Argumente für den `run-suite` Befehl in der IDT-CLI zu implementieren:

timeout-multiplier

Gibt einen Wert größer als 1,0 an, der auf alle Timeouts beim Ausführen von Tests angewendet wird.

Testläufer können dieses Argument verwenden, um das Timeout für die Testfälle zu erhöhen, die sie ausführen möchten. Wenn ein Testläufer dieses Argument in seinem `run-suite` Befehl angibt, verwendet IDT es, um den Wert der Umgebungsvariablen `IDT_TEST_TIMEOUT` zu berechnen, und legt das Feld im IDT-Kontext fest. `config.timeoutMultiplier` Um dieses Argument zu unterstützen, müssen Sie wie folgt vorgehen:

- Anstatt den Timeout-Wert direkt aus der `test.json` Datei zu verwenden, lesen Sie die Umgebungsvariable `IDT_TEST_TIMEOUT`, um den korrekt berechneten Timeout-Wert zu erhalten.
- Rufen Sie den `config.timeoutMultiplier` Wert aus dem IDT-Kontext ab und wenden Sie ihn auf Timeouts mit langer Laufzeit an.

Weitere Hinweise zum vorzeitigen Beenden aufgrund von Timeout-Ereignissen finden Sie unter.

[Geben Sie das Exit-Verhalten an](#)

stop-on-first-failure

Gibt an, dass IDT die Ausführung aller Tests beenden soll, wenn ein Fehler auftritt.

Wenn ein Testläufer dieses Argument in seinem `run-suite` Befehl angibt, beendet IDT die Ausführung von Tests, sobald ein Fehler auftritt. Wenn Testfälle jedoch parallel ausgeführt werden, kann dies zu unerwarteten Ergebnissen führen. Um Support zu implementieren, stellen Sie sicher, dass Ihre Testlogik alle laufenden Testfälle anweist, anzuhalten, temporäre Ressourcen zu bereinigen und ein Testergebnis an IDT zu melden, wenn IDT auf dieses Ereignis trifft. Weitere Informationen zum vorzeitigen Beenden von Fehlern finden Sie unter. [Geben Sie das Exit-Verhalten an](#)

group-id und test-id

Gibt an, dass IDT nur die ausgewählten Testgruppen oder Testfälle ausführen soll.

Testläufer können diese Argumente mit ihrem `run-suite` Befehl verwenden, um das folgende Verhalten bei der Testausführung anzugeben:

- Führt alle Tests innerhalb der angegebenen Testgruppen aus.
- Führt eine Auswahl von Tests innerhalb einer angegebenen Testgruppe aus.

Um diese Argumente zu unterstützen, muss die Zustandsmaschine für Ihre Testsuite einen bestimmten Satz von RunTask Choice Zuständen in Ihrer Zustandsmaschine enthalten. Wenn Sie keine benutzerdefinierte Zustandsmaschine verwenden, enthält die standardmäßige IDT-Zustandsmaschine die erforderlichen Zustände für Sie, und Sie müssen keine zusätzlichen Maßnahmen ergreifen. Wenn Sie jedoch eine benutzerdefinierte Zustandsmaschine verwenden, verwenden Sie sie [Beispiel für eine Zustandsmaschine: Führen Sie vom Benutzer ausgewählte Testgruppen aus](#) als Beispiel, um die erforderlichen Zustände zu Ihrer Zustandsmaschine hinzuzufügen.

Weitere Informationen zu IDT-CLI-Befehlen finden Sie unter [Debuggen und Ausführen benutzerdefinierter Testsuiten](#).

Schreiben Sie Ereignisprotokolle

Während der Test läuft, senden Sie Daten an `stdout` `stderr` die Konsole und schreiben dort Ereignisprotokolle und Fehlermeldungen. Hinweise zum Format von Konsolenmeldungen finden Sie unter [Nachrichtenformat der Konsole](#).

Wenn das IDT die Ausführung der Testsuite beendet hat, sind diese Informationen auch in der `test_manager.log` Datei im `<devicetester-extract-location>/results/<execution-id>/logs` Ordner verfügbar.

Sie können jeden Testfall so konfigurieren, dass die Protokolle des Testlaufs, einschließlich der Protokolle des zu testenden Geräts, in die `<group-id>_<test-id>` Datei im `<device-tester-extract-location>/results/<execution-id>/logs` Ordner geschrieben werden. Rufen Sie dazu den Pfad zur Protokolldatei aus dem IDT-Kontext mit der `testData.logFilePath` Abfrage ab, erstellen Sie eine Datei unter diesem Pfad und schreiben Sie den gewünschten Inhalt hinein. IDT aktualisiert den Pfad automatisch auf der Grundlage des laufenden Testfalls. Wenn Sie sich dafür entscheiden, die Protokolldatei für einen Testfall nicht zu erstellen, wird keine Datei für diesen Testfall generiert.

Sie können Ihre ausführbare Textdatei auch so einrichten, dass sie bei Bedarf zusätzliche Protokolldateien im `<device-tester-extract-location>/logs` Ordner erstellt. Wir empfehlen Ihnen, eindeutige Präfixe für Protokolldateinamen anzugeben, damit Ihre Dateien nicht überschrieben werden.

Ergebnisse an IDT melden

IDT schreibt Testergebnisse in die `awsiotdevicetester_report.xml` und die `suite-name_report.xml` Dateien. Diese Berichtsdateien befinden sich in `<device-tester-extract-location>/results/<execution-id>/`. Beide Berichte erfassen die Ergebnisse der Ausführung der Testsuite. Weitere Informationen zu den Schemas, die IDT für diese Berichte verwendet, finden Sie unter [Überprüfen Sie die IDT-Testergebnisse und -protokolle](#)

Um den Inhalt der `suite-name_report.xml` Datei aufzufüllen, müssen Sie den `SendResult` Befehl verwenden, um die Testergebnisse an IDT zu melden, bevor die Testausführung abgeschlossen ist. Wenn IDT die Ergebnisse eines Tests nicht finden kann, gibt es einen Fehler für den Testfall aus. Der folgende Python-Auszug zeigt die Befehle zum Senden eines Testergebnisses an IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Wenn Sie Ergebnisse nicht über die API melden, sucht IDT im Ordner mit den Testartefakten nach Testergebnissen. Der Pfad zu diesem Ordner wird in der Datei im `testData.testArtifactsPath` IDT-Kontext gespeichert. In diesem Ordner verwendet IDT die erste alphabetisch sortierte XML-Datei, die es findet, als Testergebnis.

Wenn Ihre Testlogik JUnit XML-Ergebnisse liefert, können Sie die Testergebnisse in eine XML-Datei im Ordner `artefacts` schreiben, um die Ergebnisse direkt an IDT weiterzugeben, anstatt die Ergebnisse zu analysieren und sie dann über die API an IDT zu senden.

Wenn Sie diese Methode verwenden, stellen Sie sicher, dass Ihre Testlogik die Testergebnisse korrekt zusammenfasst, und formatieren Sie Ihre Ergebnisdatei im gleichen Format wie die Datei `suite-name_report.xml`. IDT führt keine Überprüfung der von Ihnen bereitgestellten Daten durch, mit den folgenden Ausnahmen:

- IDT ignoriert alle Eigenschaften des Tags `testsuites`. Stattdessen werden die Tag-Eigenschaften anhand anderer gemeldeter Testgruppenergebnisse berechnet.
- Darin `testsuites` muss mindestens ein `testsuite` Tag vorhanden sein.

Da IDT für alle Testfälle denselben Ordner mit Artefakten verwendet und Ergebnisdateien nicht zwischen Testläufen löscht, kann diese Methode auch zu fehlerhaften Berichten führen, wenn IDT die falsche Datei liest. Es wird empfohlen, für alle Testfälle denselben Namen für die generierte

XML-Ergebnisdatei zu verwenden, um die Ergebnisse für jeden Testfall zu überschreiben und sicherzustellen, dass IDT die richtigen Ergebnisse zur Verfügung hat. Sie können in Ihrer Testsuite zwar einen gemischten Ansatz für die Berichterstattung verwenden, d. h. für einige Testfälle eine XML-Ergebnisdatei verwenden und für andere die Ergebnisse über die API einreichen, wir empfehlen diesen Ansatz jedoch nicht.

Geben Sie das Exit-Verhalten an

Konfigurieren Sie Ihre ausführbare Textdatei so, dass sie immer mit dem Exit-Code 0 beendet wird, auch wenn ein Testfall einen Fehler oder ein Fehlerergebnis meldet. Verwenden Sie Exit-Codes ungleich Null nur, um anzuzeigen, dass ein Testfall nicht ausgeführt wurde oder dass die ausführbare Testfalldatei keine Ergebnisse an IDT übermitteln konnte. Wenn IDT einen Exit-Code ungleich Null empfängt, bedeutet dies, dass der Testfall auf einen Fehler gestoßen ist, der seine Ausführung verhindert hat.

IDT kann in den folgenden Fällen anfordern oder erwarten, dass die Ausführung eines Testfalls beendet wird, bevor er abgeschlossen ist. Verwenden Sie diese Informationen, um Ihre ausführbare Testfalldatei so zu konfigurieren, dass jedes dieser Ereignisse anhand des Testfalls erkannt wird:

Timeout (Zeitüberschreitung)

Tritt auf, wenn ein Testfall länger als der in der `test.json` Datei angegebene Timeout-Wert ausgeführt wird. Wenn der Testläufer das `timeout-multiplier` Argument verwendet hat, um einen Timeout-Multiplikator anzugeben, berechnet IDT den Timeout-Wert mit dem Multiplikator.

Verwenden Sie die Umgebungsvariable `IDT_TEST_TIMEOUT`, um dieses Ereignis zu erkennen. Wenn ein Testläufer einen Test startet, setzt IDT den Wert der Umgebungsvariablen `IDT_TEST_TIMEOUT` auf den berechneten Timeout-Wert (in Sekunden) und übergibt die Variable an die ausführbare Testfalldatei. Sie können den Variablenwert lesen, um einen geeigneten Timer festzulegen.

Unterbrechen

Tritt auf, wenn der Test-Runner IDT unterbricht. Zum Beispiel durch Drücken von `Ctrl+C`

Da Terminals Signale an alle untergeordneten Prozesse weiterleiten, können Sie in Ihren Testfällen einfach einen Signal-Handler konfigurieren, um Interrupt-Signale zu erkennen.

Alternativ können Sie die API regelmäßig abfragen, um den Wert des `CancellationRequested` booleschen Werts in der API-Antwort zu überprüfen. `PollForNotifications`

Wenn IDT ein Interruptsignal empfängt, setzt es den Wert des booleschen Werts auf.
`CancellationRequested true`

Stoppt beim ersten Fehler

Tritt auf, wenn ein Testfall, der parallel zum aktuellen Testfall ausgeführt wird, fehlschlägt und der Testläufer das `stop-on-first-failure` Argument verwendet hat, um anzugeben, dass IDT beendet werden soll, wenn ein Fehler auftritt.

Um dieses Ereignis zu erkennen, können Sie die API regelmäßig abfragen, um den Wert des `CancellationRequested` booleschen Werts in der `PollForNotifications` API-Antwort zu überprüfen. Wenn IDT auf einen Fehler stößt und so konfiguriert ist, dass es beim ersten Fehler stoppt, wird der Wert des booleschen Werts `CancellationRequested` auf `true` gesetzt.

Wenn eines dieser Ereignisse eintritt, wartet IDT 5 Minuten, bis alle derzeit laufenden Testfälle abgeschlossen sind. Wenn nicht alle laufenden Testfälle innerhalb von 5 Minuten beendet werden, erzwingt IDT, jeden ihrer Prozesse zu beenden. Wenn IDT vor dem Ende der Prozesse keine Testergebnisse erhalten hat, werden die Testfälle als Timeout markiert. Als bewährte Methode sollten Sie sicherstellen, dass Ihre Testfälle die folgenden Aktionen ausführen, wenn sie auf eines der Ereignisse stoßen:

1. Beenden Sie die Ausführung der normalen Testlogik.
2. Bereinigen Sie alle temporären Ressourcen, z. B. Testartefakte, auf dem zu testenden Gerät.
3. Melden Sie IDT ein Testergebnis, z. B. einen Testfehler oder einen Fehler.
4. Beenden.

Verwenden Sie den IDT-Kontext

Wenn IDT eine Testsuite ausführt, kann die Testsuite auf einen Datensatz zugreifen, anhand dessen bestimmt werden kann, wie die einzelnen Tests ausgeführt werden. Diese Daten werden als IDT-Kontext bezeichnet. Beispielsweise wird die Benutzerdatenkonfiguration, die von Testläufern in einer `userdata.json` Datei bereitgestellt wird, Testsuiten im IDT-Kontext zur Verfügung gestellt.

Der IDT-Kontext kann als schreibgeschütztes JSON-Dokument betrachtet werden. Testsuiten können mithilfe von Standard-JSON-Datentypen wie Objekten, Arrays, Zahlen usw. Daten aus dem Kontext abrufen und Daten in den Kontext schreiben.

Kontextschema

Der IDT-Kontext verwendet das folgende Format:

```
{
  "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

Informationen aus der [config.jsonDatei](#). Das config Feld enthält außerdem das folgende zusätzliche Feld:

`config.timeoutMultiplier`

Der Multiplikator für jeden Timeout-Wert, der von der Testsuite verwendet wird. Dieser Wert wird vom Test-Runner über die IDT-CLI angegeben. Der Standardwert ist 1.

`device`

Informationen über das Gerät, das für den Testlauf ausgewählt wurde. Diese Information entspricht dem `devices` Array-Element in der [device.json](#) Datei für das ausgewählte Gerät.

`devicePool`

Informationen über den Gerätepool, der für den Testlauf ausgewählt wurde. Diese Informationen entsprechen dem Element des Gerätepool-Arrays der obersten Ebene, das in der `device.json` Datei für den ausgewählten Gerätepool definiert ist.

`resource`

Informationen zu Ressourcengeräten aus der `resource.json` Datei.

`resource.devices`

Diese Informationen entsprechen dem in der `resource.json` Datei definierten `devices` Array. Jedes `devices` Element enthält das folgende zusätzliche Feld:

`resource.device.name`

Der Name des Ressourcengeräts. Dieser Wert ist auf den `requiredResource.name` Wert in der `test.json` Datei festgelegt.

`testData.awsCredentials`

Die AWS Anmeldeinformationen, die vom Test für die Verbindung mit der AWS Cloud verwendet wurden. Diese Informationen werden aus der `config.json` Datei abgerufen.

`testData.logFilePath`

Der Pfad zur Protokolldatei, in die der Testfall Protokollnachrichten schreibt. Die Testsuite erstellt diese Datei, falls sie nicht existiert.

`userData`

Informationen, die vom Testrunner in der [userdata.json](#) Datei bereitgestellt wurden.

Greifen Sie auf Daten im Kontext zu

Sie können den Kontext mithilfe der JSONPath Notation aus Ihren JSON-Dateien und aus Ihrer ausführbaren Textdatei mit dem `GetContextValue` und abfragen `GetContextString` APIs. Die Syntax für JSONPath Zeichenketten für den Zugriff auf den IDT-Kontext variiert wie folgt:

- In `suite.json` und `test.json` verwenden `{{query}}` Sie. Das heißt, verwenden Sie nicht das Stammelement `$.`, um Ihren Ausdruck zu starten.
- In `instatemachine.json`, du benutzt `{{$.query}}`.
- In API-Befehlen verwenden Sie je nach Befehl `query` oder `{{$.query}}`. Weitere Informationen finden Sie in der Inline-Dokumentation im SDKs.

In der folgenden Tabelle werden die Operatoren in einem typischen JSONPath Ausdruck beschrieben:

Operator	Beschreibung
<code>\$</code>	Das Stammelement. Da es sich bei dem Kontextwert der obersten Ebene für IDT um ein Objekt handelt, verwenden Sie ihn normalerweise, um Ihre <code>\$.</code> Abfragen zu starten.
<code>.childName</code>	Greift auf das untergeordnete Element mit dem Namen eines Objekts <code>childName</code> zu. Wenn es auf ein Array angewendet wird, ergibt dies ein neues Array, bei dem dieser Operator auf jedes Element angewendet wird. Beim Elementnamen wird zwischen Groß- und Kleinschreibung unterschieden. Die Abfrage für den Zugriff auf den <code>awsRegion</code> Wert im <code>config</code> Objekt lautet beispielsweise <code>\$.config.awsRegion</code> .
<code>[start:end]</code>	Filtert Elemente aus einem Array und ruft Elemente ab, die mit dem <code>start</code> Index beginnen und bis zum <code>end</code> Index aufsteigen, jeweils inklusive.

Operator	Beschreibung
[index1, index2, ... , indexN]	Filtert Elemente aus einem Array und ruft nur Elemente aus den angegebenen Indizes ab.
[?(expr)]	Filtert Elemente aus einem Array mithilfe des expr Ausdrucks. Dieser Ausdruck muss einen booleschen Wert ergeben.

Verwenden Sie die folgende Syntax, um Filterausdrücke zu erstellen:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

In dieser Syntax gilt:

- `jsonpath` ist ein JSONPath, das die Standard-JSON-Syntax verwendet.
- `value` ist ein beliebiger benutzerdefinierter Wert, der die Standard-JSON-Syntax verwendet.
- `operator` ist einer der folgenden Operatoren:
 - `<` (Weniger als)
 - `<=` (Weniger als oder gleich)
 - `==` (Gleich)

Wenn der Wert JSONPath oder in Ihrem Ausdruck ein Array, ein boolescher Wert oder ein Objektwert ist, dann ist dies der einzige unterstützte binäre Operator, den Sie verwenden können.

- `>=` (Größer als oder gleich)
- `>` (Größer als)
- `=~` (Übereinstimmung mit regulären Ausdrücken). Um diesen Operator in einem Filterausdruck zu verwenden, muss der Wert JSONPath oder auf der linken Seite des Ausdrucks eine Zeichenfolge ergeben, und auf der rechten Seite muss es sich um einen Musterwert handeln, der der [RE2Syntax](#) folgt.

Sie können JSONPath Abfragen in der Form `{{query}}` als Platzhalterzeichenfolgen innerhalb der `environmentVariables` Felder `args` und in `test.json` Dateien und innerhalb der `environmentVariables` Felder in `suite.json` Dateien verwenden. IDT führt eine Kontextsuche

durch und füllt die Felder mit dem ausgewerteten Wert der Abfrage auf. In der `suite.json` Datei können Sie beispielsweise Platzhalterzeichenfolgen verwenden, um Umgebungsvariablenwerte anzugeben, die sich mit jedem Testfall ändern, und IDT füllt die Umgebungsvariablen mit dem richtigen Wert für jeden Testfall. Wenn Sie jedoch Platzhalterzeichenfolgen in `test.json` und `suite.json` -Dateien verwenden, gelten für Ihre Abfragen die folgenden Überlegungen:

- Sie müssen jedes Vorkommen des `devicePool` Schlüssels in Ihrer Abfrage in Kleinbuchstaben angeben. Das heißt, verwenden Sie `devicepool` stattdessen.
- Für Arrays können Sie nur Zeichenketten-Arrays verwenden. Darüber hinaus verwenden Arrays ein nicht standardmäßiges Format. `item1, item2, ..., itemN` Wenn das Array nur ein Element enthält, wird es als `serialisiertitem`, sodass es nicht von einem Zeichenkettenfeld zu unterscheiden ist.
- Sie können keine Platzhalter verwenden, um Objekte aus dem Kontext abzurufen.

Aus diesen Gründen empfehlen wir, wann immer möglich, die API für den Zugriff auf den Kontext in Ihrer Testlogik anstelle von Platzhalterzeichenfolgen in `test.json` und `suite.json` -Dateien zu verwenden. In einigen Fällen kann es jedoch praktischer sein, JSONPath Platzhalter zu verwenden, um einzelne Zeichenketten abzurufen, die als Umgebungsvariablen festgelegt werden sollen.

Einstellungen für Testläufer konfigurieren

Um benutzerdefinierte Testsuiten auszuführen, müssen Testläufer ihre Einstellungen auf der Grundlage der Testsuite konfigurieren, die sie ausführen möchten. Die Einstellungen werden auf der Grundlage von JSON-Konfigurationsdateivorlagen angegeben, die sich im `<device-tester-extract-location>/configs/` Ordner befinden. Falls erforderlich, müssen Testläufer auch AWS Anmeldeinformationen einrichten, die IDT für die Verbindung mit der AWS Cloud verwendet.

Als Testautor müssen Sie diese Dateien konfigurieren, um [Ihre Testsuite zu debuggen](#). Sie müssen den Testläufern Anweisungen geben, damit sie die folgenden Einstellungen nach Bedarf für die Ausführung Ihrer Testsuiten konfigurieren können.

Konfigurieren von `device.json`

Die `device.json` Datei enthält Informationen über die Geräte, auf denen die Tests ausgeführt werden (z. B. IP-Adresse, Anmeldeinformationen, Betriebssystem und CPU-Architektur).

Testläufer können diese Informationen mithilfe der folgenden `device.json` Vorlagendatei bereitstellen, die sich im `<device-tester-extract-location>/configs/` Ordner befindet.

```
[
  {
    "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>",
      }
    ]
  }
]
```

```
    }  
  ]  
}  
]
```

Nachfolgend sind alle Pflichtfelder beschrieben:

id

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

sku

Ein alphanumerischer Wert, durch den das zu testende Gerät eindeutig identifiziert wird. Die SKU wird verwendet, um qualifizierte Geräte nachzuverfolgen.

Note

Wenn du dein Motherboard im AWS Partner Gerätecatalog auflisten möchtest, muss die hier angegebene SKU mit der SKU übereinstimmen, die du bei der Angebotserstellung verwendest.

features

Optional. Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Gerätefunktionen sind benutzerdefinierte Werte, die Sie in Ihrer Testsuite konfigurieren. Sie müssen Ihren Testläufern Informationen über die Namen und Werte der Funktionen zur Verfügung stellen, die in die `device.json` Datei aufgenommen werden sollen. Wenn Sie beispielsweise ein Gerät testen möchten, das als MQTT-Server für andere Geräte fungiert, können Sie Ihre Testlogik so konfigurieren, dass bestimmte unterstützte Stufen für ein Feature mit dem Namen `MQTT_QOS` validiert werden. Testläufer geben diesen Feature-Namen an und setzen den Feature-Wert auf die QOS-Stufen, die von ihrem Gerät unterstützt werden. Sie können die bereitgestellten Informationen mit der Abfrage aus dem [IDT-Kontext](#) oder mit der `devicePool.features` Abfrage aus dem [State-Machine-Kontext](#) abrufen. `pool.features`

`features.name`

Der Name des Features.

`features.value`

Die unterstützten Feature-Werte.

`features.configs`

Konfigurationseinstellungen für die Funktion, falls erforderlich.

`features.config.name`

Der Name der Konfigurationseinstellung.

`features.config.value`

Die unterstützten Einstellungswerte.

`devices`

Eine Reihe von Geräten im Pool, die getestet werden sollen. Es ist mindestens ein Gerät erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur Werte `uart` für physische Geräte `ssh` und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, an die das Gerät angeschlossen ist.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `uart` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers gegenüber dem Benutzer innerhalb des Containers. Der Standardwert ist der im Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

Note

Um zu überprüfen, ob Testläufer die falsche Geräteverbindung für einen Test konfigurieren, können Sie Daten `pool.Devices[0].Connectivity.Protocol` aus dem Zustandsmaschinenkontext abrufen und ihn mit dem erwarteten Wert in einem Choice Status vergleichen. Wenn ein falsches Protokoll verwendet wird, drucken Sie eine Nachricht mit dem LogMessage Status und wechseln Sie zum Fail Status. Alternativ können Sie den Fehlerbehandlungscode verwenden, um einen Testfehler für falsche Gerätetypen zu melden.

(Optional) Konfigurieren Sie `userdata.json`

Die `userdata.json` Datei enthält alle zusätzlichen Informationen, die für eine Testsuite erforderlich sind, aber nicht in der Datei angegeben sind. `device.json` Das Format dieser Datei hängt von der [userdata_scheme.json](#) Datei ab, die in der Testsuite definiert ist. Wenn Sie ein Testautor sind, stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Testsuiten ausführen.

(Optional) Konfigurieren Sie `resource.json`

Die `resource.json` Datei enthält Informationen zu allen Geräten, die als Ressourcengeräte verwendet werden. Ressourcengeräte sind Geräte, die zum Testen bestimmter Funktionen eines zu testenden Geräts erforderlich sind. Um beispielsweise die Bluetooth-Fähigkeit eines Geräts zu testen, können Sie ein Ressourcengerät verwenden, um zu testen, ob Ihr Gerät erfolgreich eine

Verbindung zu dem Gerät herstellen kann. Ressourcengeräte sind optional, und Sie können so viele Ressourcengeräte benötigen, wie Sie benötigen. Als Testautor verwenden Sie die [Datei test.json](#), um die Funktionen der Ressourcengeräte zu definieren, die für einen Test erforderlich sind. Testläufer verwenden dann die `resource.json` Datei, um einen Pool von Ressourcengeräten bereitzustellen, die über die erforderlichen Funktionen verfügen. Stellen Sie sicher, dass Sie diese Informationen Benutzern zur Verfügung stellen, die die von Ihnen geschriebenen Testsuiten ausführen werden.

Testläufer können diese Informationen mithilfe der folgenden `resource.json` Vorlagendatei bereitstellen, die sich im `<device-tester-extract-location>/configs/` Ordner befindet.

```
[
  {
    "id": "<pool-id>",
    "features": [
      {
        "name": "<feature-name>",
        "version": "<feature-value>",
        "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>",
      }
    }
  ]
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

id

Eine benutzerdefinierte alphanumerische ID, die eine Sammlung von Geräten, den sogenannten Gerätepool, eindeutig identifiziert. Geräte, die zu einem Pool gehören, müssen über identische Hardware verfügen. Bei der Ausführung einer Reihe von Tests werden Geräte im Pool verwendet, um die Workload zu parallelisieren. Mehrere Geräte werden verwendet, um verschiedene Tests auszuführen.

features

Optional. Ein Array, das die Funktionen enthält, die das Gerät unterstützt. Die in diesem Feld erforderlichen Informationen sind in den [test.json-Dateien](#) in der Testsuite definiert und bestimmen, welche Tests ausgeführt werden und wie diese Tests ausgeführt werden. Wenn die Testsuite keine Funktionen benötigt, ist dieses Feld nicht erforderlich.

features.name

Der Name der Funktion.

features.version

Die Feature-Version.

features.jobSlots

Einstellung, die angibt, wie viele Tests das Gerät gleichzeitig verwenden können. Der Standardwert ist 1.

devices

Eine Reihe von Geräten im Pool, die getestet werden sollen. Es ist mindestens ein Gerät erforderlich.

`devices.id`

Eine benutzerdefinierte eindeutige Kennung für das zu testende Gerät.

`connectivity.protocol`

Das Kommunikationsprotokoll, das für die Kommunikation mit diesem Gerät verwendet wird. Jedes Gerät in einem Pool muss dasselbe Protokoll verwenden.

Derzeit werden nur Werte `uart` für physische Geräte `ssh` und `docker` für Docker-Container unterstützt.

`connectivity.ip`

Die IP-Adresse des zu testenden Geräts.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.port`

Optional. Die Portnummer, die für SSH-Verbindungen verwendet werden soll.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth`

Authentifizierungsinformationen für die Verbindung.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `ssh` festgelegt ist.

`connectivity.auth.method`

Die Authentifizierungsmethode, die für den Zugriff über ein bestimmtes Verbindungsprotokoll auf ein Gerät verwendet wird.

Unterstützte Werte sind:

- `pki`
- `password`

`connectivity.auth.credentials`

Die für die Authentifizierung verwendeten Anmeldeinformationen.

`connectivity.auth.credentials.password`

Das Passwort für die Anmeldung am Gerät wird überprüft.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `password` festgelegt ist.

`connectivity.auth.credentials.privKeyPath`

Der vollständige Pfad zum privaten Schlüssel, der für die Anmeldung bei dem zu testenden Gerät verwendet wird.

Dieser Wert gilt nur, wenn `connectivity.auth.method` auf `pki` festgelegt ist.

`connectivity.auth.credentials.user`

Der Benutzername für die Anmeldung bei dem zu testenden Gerät.

`connectivity.serialPort`

Optional. Die serielle Schnittstelle, an die das Gerät angeschlossen ist.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `uart` festgelegt ist.

`connectivity.containerId`

Die Container-ID oder der Name des getesteten Docker-Containers.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

`connectivity.containerUser`

Optional. Der Name des Benutzers gegenüber dem Benutzer innerhalb des Containers. Der Standardwert ist der im Dockerfile angegebene Benutzer.

Der Standardwert ist 22.

Diese Eigenschaft gilt nur, wenn `connectivity.protocol` auf `docker` festgelegt ist.

(Optional) Konfigurieren Sie `config.json`

Die `config.json` Datei enthält Konfigurationsinformationen für IDT. In der Regel müssen Testläufer diese Datei nicht ändern, es sei denn, sie müssen ihre AWS Benutzeranmeldeinformationen für IDT und optional eine AWS Region angeben. Wenn AWS Anmeldeinformationen mit den erforderlichen

Berechtigungen bereitgestellt werden, sammelt AWS IoT Device Tester Nutzungsdaten und sendet sie an. AWS Dies ist eine Opt-in-Funktion, die zur Verbesserung der IDT-Funktionalität verwendet wird. Weitere Informationen finden Sie unter [IDT-Nutzungsmetriken](#).

Testläufer können ihre AWS Anmeldeinformationen auf eine der folgenden Arten konfigurieren:

- Anmeldeinformationsdatei

IDT verwendet die gleiche Anmeldeinformationsdatei wie das AWS CLI. Weitere Informationen finden Sie unter [Konfigurations- und Anmeldeinformationsdateien](#).

Der Speicherort der Datei mit den Anmeldeinformationen variiert je nach verwendetem Betriebssystem:

- macOS Linux: `~/.aws/credentials`
- Windows: `C:\Users\UserName\.aws\credentials`
- Umgebungsvariablen

Umgebungsvariablen sind Variablen, die vom Betriebssystem gepflegt und von Systembefehlen verwendet werden. Variablen, die während einer SSH-Sitzung definiert wurden, sind nach dem Schließen dieser Sitzung nicht verfügbar. IDT kann die `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen `AWS_ACCESS_KEY_ID` und zum Speichern von Anmeldeinformationen verwenden `AWS`

Um diese Variablen auf Linux, macOS oder Unix festzulegen, verwenden Sie `export`:

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

In Windows können Sie die Variablen mit `set` festlegen:

```
set AWS_ACCESS_KEY_ID=<your_access_key_id>
set AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

Um AWS Anmeldeinformationen für IDT zu konfigurieren, bearbeiten Testläufer den `auth` Abschnitt in der `config.json` Datei, die sich im `<device-tester-extract-location>/configs/` Ordner befindet.

```
{
```

```
"log": {
  "location": "logs"
},
"configFiles": {
  "root": "configs",
  "device": "configs/device.json"
},
"testPath": "tests",
"reportPath": "results",
"awsRegion": "<region>",
"auth": {
  "method": "file | environment",
  "credentials": {
    "profile": "<profile-name>"
  }
}
}
```

Nachfolgend sind alle Pflichtfelder beschrieben:

 Note

Alle Pfade in dieser Datei sind relativ zu definiert. *<device-tester-extract-location>*

log.location

Der Pfad zum Protokollordner in der *<device-tester-extract-location>*.

configFiles.root

Der Pfad zu dem Ordner, der die Konfigurationsdateien enthält.

configFiles.device

Der Pfad zur *device.json* Datei.

testPath

Der Pfad zu dem Ordner, der Testsuiten enthält.

reportPath

Der Pfad zu dem Ordner, der Testergebnisse enthält, nachdem IDT eine Testsuite ausgeführt hat.

awsRegion

Optional. Die AWS Region, die die Testsuiten verwenden werden. Wenn nicht festgelegt, verwenden Testsuiten die in jeder Testsuite angegebene Standardregion.

auth.method

Die Methode, die IDT zum Abrufen von AWS Anmeldeinformationen verwendet. Unterstützte Werte sind `file` das Abrufen von Anmeldeinformationen aus einer Anmeldeinformationsdatei und `environment` das Abrufen von Anmeldeinformationen mithilfe von Umgebungsvariablen.

auth.credentials.profile

Das zu verwendende Anmeldeinformationsprofil aus der Anmeldeinformationsdatei. Diese Eigenschaft gilt nur, wenn `auth.method` auf `file` festgelegt ist.

Debuggen und Ausführen benutzerdefinierter Testsuiten

Nachdem die [erforderliche Konfiguration](#) festgelegt wurde, kann IDT Ihre Testsuite ausführen. Die Laufzeit der vollständigen Testsuite hängt von der Hardware und der Zusammensetzung der Testsuite ab. Als Referenz: Es dauert ungefähr 30 Minuten, bis die vollständige AWS IoT Greengrass Qualifizierungstestsuite auf einem Raspberry Pi 3B abgeschlossen ist.

Während Sie Ihre Testsuite schreiben, können Sie IDT verwenden, um die Testsuite im Debug-Modus auszuführen, um Ihren Code vor der Ausführung zu überprüfen oder ihn Testläufern zur Verfügung zu stellen.

Führen Sie IDT im Debug-Modus aus

Da Testsuiten auf IDT angewiesen sind, um mit Geräten zu interagieren, den Kontext bereitzustellen und Ergebnisse zu erhalten, können Sie Ihre Testsuiten nicht einfach ohne IDT-Interaktion in einer IDE debuggen. Zu diesem Zweck stellt die IDT-CLI den `debug-test-suite` Befehl bereit, mit dem Sie IDT im Debug-Modus ausführen können. Führen Sie den folgenden Befehl aus, um die verfügbaren Optionen für `debug-test-suite` anzuzeigen:

```
devicetester_[linux | mac | win_x86-64] debug-test-suite -h
```

Wenn Sie IDT im Debug-Modus ausführen, startet IDT weder die Testsuite noch die Zustandsmaschine. Stattdessen interagiert IDT mit Ihrer IDE, um auf Anfragen von der in der IDE ausgeführten Testsuite zu antworten und die Protokolle auf der Konsole auszudrucken. IDT hat kein Timeout und wartet mit dem Beenden, bis es manuell unterbrochen wird. Im Debug-Modus führt

IDT die Zustandsmaschine auch nicht aus und generiert keine Berichtsdateien. Um Ihre Testsuite zu debuggen, müssen Sie Ihre IDE verwenden, um einige Informationen bereitzustellen, die IDT normalerweise aus den JSON-Konfigurationsdateien bezieht. Stellen Sie sicher, dass Sie die folgenden Informationen angeben:

- Umgebungsvariablen und Argumente für jeden Test. IDT liest diese Informationen nicht von `test.json` oder `suite.json`.
- Argumente zur Auswahl von Ressourcengeräten. IDT liest diese Informationen nicht aus `test.json`.

Gehen Sie wie folgt vor, um Ihre Testsuiten zu debuggen:

1. Erstellen Sie die Einstellungskonfigurationsdateien, die für die Ausführung der Testsuite erforderlich sind. Wenn Ihre Testsuite beispielsweise `device.json`, und erfordert `resource.json`, stellen Sie sicher `user_data.json`, dass Sie sie alle nach Bedarf konfigurieren.
2. Führen Sie den folgenden Befehl aus, um IDT in den Debug-Modus zu versetzen und alle Geräte auszuwählen, die für die Ausführung des Tests erforderlich sind.

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

Nachdem Sie diesen Befehl ausgeführt haben, wartet IDT auf Anfragen von der Testsuite und beantwortet sie dann. IDT generiert auch die Umgebungsvariablen, die für den Fallprozess für das IDT Client SDK erforderlich sind.

3. Verwenden Sie in Ihrer IDE die debug Konfiguration `run` oder, um Folgendes zu tun:
 - a. Legen Sie die Werte der IDT-generierten Umgebungsvariablen fest.
 - b. Legen Sie den Wert aller Umgebungsvariablen oder Argumente fest, die Sie in Ihrer `test.json` AND-Datei angegeben haben. `suite.json`
 - c. Legen Sie nach Bedarf Haltepunkte fest.
4. Führen Sie die Testsuite in Ihrer IDE aus.

Sie können die Testsuite so oft wie nötig debuggen und erneut ausführen. IDT tritt im Debug-Modus nicht auf.

5. Nachdem Sie das Debuggen abgeschlossen haben, unterbrechen Sie IDT, um den Debug-Modus zu verlassen.

IDT-CLI-Befehle zum Ausführen von Tests

Im folgenden Abschnitt werden die IDT-CLI-Befehle beschrieben:

IDT v4.0.0

`help`

Listet Informationen über den angegebenen Befehl auf.

`list-groups`

Listet die Gruppen in der jeweiligen Testsuite auf.

`list-suites`

Listet die verfügbaren Testsuites auf.

`list-supported-products`

Listet die unterstützten Produkte für Ihre Version von IDT auf, in diesem Fall AWS IoT Greengrass Versionen, und die Versionen der AWS IoT Greengrass Qualifizierungstestsuite, die für die aktuelle IDT-Version verfügbar sind.

`list-test-cases`

Listet die Testfälle in einer bestimmten Testgruppe auf. Die folgende Option wird unterstützt:

- `group-id`. Die Testgruppe, nach der gesucht werden soll. Diese Option ist erforderlich und muss eine einzelne Gruppe angeben.

`run-suite`

Führt eine Reihe von Tests in einem Pool von Geräten aus. Im Folgenden sind einige häufig verwendete Optionen aufgeführt:

- `suite-id`. Die auszuführende Version der Testsuite. Wenn nicht angegeben, verwendet IDT die neueste Version im `tests`-Ordner.
- `group-id`. Die auszuführenden Testgruppen als kommasetrennte Liste. Bei fehlender Angabe führt IDT alle Testgruppen in der Testsuite aus.
- `test-id`. Die auszuführenden Testfälle als kommasetrennte Liste. Wenn angegeben, muss `group-id` eine einzelne Gruppe angeben.
- `pool-id`. Der zu testende Gerätepool. Testläufer müssen einen Pool angeben, wenn in Ihrer `device.json` Datei mehrere Gerätepools definiert sind.

- `timeout-multiplier`. Konfiguriert IDT so, dass das in der `test.json` Datei angegebene Timeout für die Testausführung für einen Test mit einem benutzerdefinierten Multiplikator geändert wird.
- `stop-on-first-failure`. Konfiguriert IDT so, dass die Ausführung beim ersten Fehler gestoppt wird. Diese Option sollte mit `group-id` verwendet werden, um die angegebenen Testgruppen zu debuggen.
- `userdata`. Legt die Datei fest, die Benutzerdateninformationen enthält, die zum Ausführen der Testsuite erforderlich sind. Dies ist nur erforderlich, wenn `userdataRequired` es in der `suite.json` Datei für die Testsuite auf `true` gesetzt ist.

Weitere Informationen zu `run-suite`-Optionen erhalten Sie mit der `help`-Option:

```
devicetester_[linux | mac | win_x86-64] run-suite -h
```

debug-test-suite

Führen Sie die Testsuite im Debug-Modus aus. Weitere Informationen finden Sie unter [Führen Sie IDT im Debug-Modus aus](#).

Überprüfen Sie die IDT-Testergebnisse und -protokolle

In diesem Abschnitt wird das Format beschrieben, in dem IDT Konsolenprotokolle und Testberichte generiert.

Nachrichtenformat der Konsole

AWS IoT Device Tester verwendet ein Standardformat für das Drucken von Nachrichten an die Konsole, wenn eine Testsuite gestartet wird. Der folgende Auszug zeigt ein Beispiel für eine von IDT generierte Konsolennachricht.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0  
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Die meisten Konsolennachrichten bestehen aus den folgenden Feldern:

`time`

Ein vollständiger ISO 8601-Zeitstempel für das protokollierte Ereignis.

level

Die Nachrichtenebene für das protokollierte Ereignis. In der Regel ist die Ebene der protokollierten Nachricht eine von `info`, `warn`, `error`. IDT gibt eine `fatal panic` OR-Nachricht aus, wenn es auf ein erwartetes Ereignis trifft, das dazu führt, dass es vorzeitig beendet wird.

msg

Die protokollierte Nachricht.

executionId

Eine eindeutige ID-Zeichenfolge für den aktuellen IDT-Prozess. Diese ID wird verwendet, um zwischen einzelnen IDT-Läufen zu unterscheiden.

Von einer Testsuite generierte Konsolennachrichten enthalten zusätzliche Informationen über das zu testende Gerät und die Testsuite, Testgruppe und Testfälle, die IDT ausführt. Der folgende Auszug zeigt ein Beispiel für eine Konsolennachricht, die aus einer Testsuite generiert wurde.

```
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
```

Der für die Testsuite spezifische Teil der Konsolennachricht enthält die folgenden Felder:

suiteId

Der Name der Testsuite, die gerade läuft.

groupId

Die ID der Testgruppe, die gerade läuft.

testCaseId

Die ID des aktuell laufenden Testfalls.

deviceId

Eine ID des zu testenden Geräts, das der aktuelle Testfall verwendet.

Um eine Testzusammenfassung auf der Konsole zu drucken, wenn ein IDT die Ausführung eines Tests abgeschlossen hat, müssen Sie einen [ReportStatus](#) in Ihre Zustandsmaschine aufnehmen.

Die Testzusammenfassung enthält Informationen über die Testsuite, die Testergebnisse für jede Gruppe, die ausgeführt wurde, und die Speicherorte der generierten Protokolle und Berichtsdateien. Das folgende Beispiel zeigt eine Meldung mit einer Testzusammenfassung.

```

===== 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 Berichtsschema von Device Tester

`awsiotdevicetester_report.xml` ist ein signierter Bericht, der die folgenden Informationen enthält:

- Die IDT-Version.
- Die Version der Testsuite.
- Die Berichtssignatur und der Schlüssel, die zum Signieren des Berichts verwendet wurden.
- Die Geräte-SKU und der Name des Gerätepools, die in der `device.json` Datei angegeben sind.
- Die Produktversion und die getesteten Gerätefunktionen.
- Die aggregierte Zusammenfassung der Testergebnisse. Diese Informationen entsprechen denen, die in der `suite-name_report.xml` Datei enthalten sind.

```

<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>

```

Die Datei `awsiotdevicetester_report.xml` enthält ein `<awsproduct>`-Tag mit Informationen zum getesteten Produkt und den Produktfunktionen, die nach einer Reihe von Tests validiert wurden.

Im `<awsproduct>`-Tag verwendete Attribute

name

Der Name des getesteten Produkts.

version

Die Version des getesteten Produkts.

features

Die validierten Funktionen Als markierte Funktionen `required` sind erforderlich, damit die Testsuite das Gerät validieren kann. Der folgende Ausschnitt zeigt, wie diese Informationen in der Datei `awsiotdevicetester_report.xml` angezeigt werden.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Als markierte Funktionen `optional` sind für die Validierung nicht erforderlich. Die folgenden Codeausschnitte zeigen optionale Funktionen.

```
<feature name="hsi" value="supported" type="optional"></feature>
```

```
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Berichtsschema der Testsuite

Der `suite-name_Result.xml` Bericht ist im [JUnit XML-Format](#). Sie können ihn in Continuous Integration and Deployment-Plattformen wie [Jenkins](#), [Bamboo](#) usw. integrieren. Der Bericht enthält eine Zusammenfassung der Testergebnisse.

```
<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>
```

Im Berichtsabschnitt sowohl im `awsiotdevicetester_report.xml` als auch im Abschnitt *suite-name_report.xml* werden die durchgeführten Tests und die Ergebnisse aufgeführt.

Im ersten XML-Tag `<testsuites>` ist die Zusammenfassung der Testausführung enthalten. Zum Beispiel:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
  disabled="0">
```

Im **<testsuites>**-Tag verwendete Attribute

name

Name der Testsuite

time

Die Zeit in Sekunden, die für die Ausführung der Testsuite benötigt wurde.

tests

Die Anzahl der ausgeführten Tests.

failures

Die Anzahl der ausgeführten Tests, die den Test nicht bestanden haben

errors

Die Anzahl der Tests, die IDT nicht ausführen konnte.

disabled

Dieses Attribut wird nicht verwendet und kann ignoriert werden.

Falls bei Tests Fehler auftreten, können Sie den fehlgeschlagenen Test identifizieren, indem Sie die XML-Tags von `<testsuites>` überprüfen. Die XML-Tags von `<testsuite>` im `<testsuites>`-Tag zeigen die Ergebniszusammenfassung eines Tests für eine Testgruppe. Zum Beispiel:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
errors="0" skipped="0">
```

Das Format ähnelt dem `<testsuites>`-Tag, weist aber das Attribut `skipped` auf, das nicht verwendet wird und ignoriert werden kann. Innerhalb der einzelnen `<testsuite>`-XML-Tags befinden sich `<testcase>`-Tags für alle ausgeführten Tests einer Testgruppe. Zum Beispiel:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Im `<testcase>`-Tag verwendete Attribute

`name`

Der Name des Tests

`attempts`

Gibt an, wie oft IDT den Testfall ausgeführt hat.

Wenn ein Testfall fehlschlägt oder ein Fehler auftritt, werden `<failure>`- oder `<error>`-Tags hinzugefügt, um das `<testcase>`-Tag mit Informationen für die Fehlerbehebung zu versehen. Zum Beispiel:

```
<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-Nutzungsmetriken

Wenn Sie AWS Anmeldeinformationen mit den erforderlichen Berechtigungen angeben, sammelt AWS IoT Device Tester Nutzungsdaten und sendet sie an AWS. Dies ist eine Opt-in-Funktion, die zur Verbesserung der IDT-Funktionalität verwendet wird. IDT sammelt Informationen wie die folgenden:

- Die AWS-Konto ID, die zur Ausführung von IDT verwendet wurde
- Die IDT-CLI-Befehle, die zum Ausführen von Tests verwendet werden

- Die Testsuite, die ausgeführt wird
- Die Testsuiten im `<device-tester-extract-location>` Ordner
- Die Anzahl der im Gerätepool konfigurierten Geräte
- Testfallnamen und Laufzeiten
- Informationen zu den Testergebnissen, z. B. ob Tests bestanden oder fehlgeschlagen sind, ob Fehler aufgetreten sind oder ob sie übersprungen wurden
- Getestete Produktmerkmale
- Verhalten beim Beenden von IDT, z. B. unerwartete oder vorzeitige Austritte

Alle Informationen, die IDT sendet, werden auch in einer `metrics.log` Datei im Ordner protokolliert. `<device-tester-extract-location>/results/<execution-id>/` In der Protokolldatei können Sie die Informationen einsehen, die während eines Testlaufs gesammelt wurden. Diese Datei wird nur generiert, wenn Sie Nutzungsmetriken erfassen möchten.

Um die Erfassung von Messwerten zu deaktivieren, müssen Sie keine zusätzlichen Maßnahmen ergreifen. Speichern Sie Ihre AWS Anmeldeinformationen einfach nicht, und wenn Sie AWS Anmeldeinformationen gespeichert haben, konfigurieren Sie die `config.json`-Datei nicht für den Zugriff darauf.

Konfigurieren Sie Ihre AWS Anmeldeinformationen

Wenn Sie noch keine haben AWS-Konto, müssen Sie [eine erstellen](#). Wenn Sie bereits über eine verfügen AWS-Konto, müssen Sie lediglich [die erforderlichen Berechtigungen für Ihr Konto konfigurieren](#), damit IDT in Ihrem Namen Nutzungsdaten AWS an diese senden kann.

Schritt 1: Erstellen Sie ein AWS-Konto

In diesem Schritt erstellen und konfigurieren Sie eine AWS-Konto. Wenn Sie bereits eine haben AWS-Konto, fahren Sie mit fort [the section called “Schritt 2: Konfigurieren von Berechtigungen für IDT”](#).

Melde dich für eine an AWS-Konto

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie <https://portal.aws.amazon.com/billing/die-Anmeldung>.

2. Folgen Sie den Online-Anweisungen.

Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontos wird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um [Aufgaben auszuführen, die Root-Benutzerzugriff erfordern](#).

AWS sendet Ihnen nach Abschluss des Anmeldevorgangs eine Bestätigungs-E-Mail. Du kannst jederzeit deine aktuellen Kontoaktivitäten einsehen und dein Konto verwalten, indem du zu <https://aws.amazon.com/> gehst und Mein Konto auswählst.

Erstellen eines Benutzers mit Administratorzugriff

Nachdem Sie sich für einen angemeldet haben AWS-Konto, sichern Sie Ihren Root-Benutzer des AWS-Kontos AWS IAM Identity Center, aktivieren und erstellen Sie einen Administratorbenutzer, sodass Sie den Root-Benutzer nicht für alltägliche Aufgaben verwenden.

Sichern Sie Ihre Root-Benutzer des AWS-Kontos

1. Melden Sie sich [AWS-Managementkonsole](#) als Kontoinhaber an, indem Sie Root-Benutzer auswählen und Ihre AWS-Konto E-Mail-Adresse eingeben. Geben Sie auf der nächsten Seite Ihr Passwort ein.

Hilfe bei der Anmeldung mit dem Root-Benutzer finden Sie unter [Anmelden als Root-Benutzer](#) im AWS-Anmeldung -Benutzerhandbuch zu.

2. Aktivieren Sie die Multi-Faktor-Authentifizierung (MFA) für den Root-Benutzer.

Anweisungen finden Sie unter [Aktivieren eines virtuellen MFA-Geräts für Ihren AWS-Konto Root-Benutzer \(Konsole\)](#) im IAM-Benutzerhandbuch.

Erstellen eines Benutzers mit Administratorzugriff

1. Aktivieren Sie das IAM Identity Center.

Anweisungen finden Sie unter [Aktivieren AWS IAM Identity Center](#) im AWS IAM Identity Center - Benutzerhandbuch.

2. Gewähren Sie einem Administratorbenutzer im IAM Identity Center Benutzerzugriff.

Ein Tutorial zur Verwendung von IAM-Identity-Center-Verzeichnis als Identitätsquelle finden Sie IAM-Identity-Center-Verzeichnis im Benutzerhandbuch unter [Benutzerzugriff mit der Standardeinstellung konfigurieren](#).AWS IAM Identity Center

Anmelden als Administratorbenutzer

- Um sich mit Ihrem IAM-Identity-Center-Benutzer anzumelden, verwenden Sie die Anmelde-URL, die an Ihre E-Mail-Adresse gesendet wurde, als Sie den IAM-Identity-Center-Benutzer erstellt haben.

Hilfe bei der Anmeldung mit einem IAM Identity Center-Benutzer finden Sie [im AWS-Anmeldung Benutzerhandbuch unter Anmeldung beim AWS Access-Portal](#).

Weiteren Benutzern Zugriff zuweisen

1. Erstellen Sie im IAM-Identity-Center einen Berechtigungssatz, der den bewährten Vorgehensweisen für die Anwendung von geringsten Berechtigungen folgt.

Anweisungen hierzu finden Sie unter [Berechtigungssatz erstellen](#) im AWS IAM Identity Center - Benutzerhandbuch.

2. Weisen Sie Benutzer einer Gruppe zu und weisen Sie der Gruppe dann Single Sign-On-Zugriff zu.

Eine genaue Anleitung finden Sie unter [Gruppen hinzufügen](#) im AWS IAM Identity Center - Benutzerhandbuch.

Schritt 2: Konfigurieren von Berechtigungen für IDT

In diesem Schritt konfigurieren Sie die Berechtigungen, die IDT zum Ausführen von Tests und zum Sammeln von IDT-Nutzungsdaten verwendet. Sie können das AWS-Managementkonsole oder AWS Command Line Interface (AWS CLI) verwenden, um eine IAM-Richtlinie und einen Benutzer für IDT zu erstellen und dann Richtlinien an den Benutzer anzuhängen.

- [So konfigurieren Sie Berechtigungen für IDT \(Konsole\)](#):
- [So konfigurieren Sie Berechtigungen für IDT \(AWS CLI\)](#):

So konfigurieren Sie Berechtigungen für IDT (Konsole)

Gehen Sie folgendermaßen vor, um mithilfe der Konsole Berechtigungen für IDT für AWS IoT Greengrass zu konfigurieren.

1. Melden Sie sich bei der [IAM-Konsole](#) an.
2. Erstellen Sie eine vom Kunden verwaltete Richtlinie, die Berechtigungen zum Erstellen von Rollen mit bestimmten Berechtigungen erteilt.
 - a. Wählen Sie im Navigationsbereich Richtlinien und dann Richtlinie erstellen.
 - b. Ersetzen Sie auf der Registerkarte JSON den Platzhalterinhalt durch die folgende Richtlinie.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot-device-tester:SendMetrics"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. Wählen Sie Next: Tags (Weiter: Tags) aus.
 - d. Klicken Sie auf Weiter: Prüfen.
 - e. Geben Sie unter Name **IDTUsageMetricsIAMPermissions** ein. Überprüfen Sie unter Summary (Zusammenfassung) die von Ihrer Richtlinie gewährten Berechtigungen.
 - f. Wählen Sie Richtlinie erstellen aus.
3. Erstellen Sie einen IAM-Benutzer und weisen Sie dem Benutzer Berechtigungen zu.
 - a. Erstellen Sie einen IAM-Benutzer. Folgen Sie den Schritten 1 bis 5 unter [Erstellen von IAM-Benutzern \(Konsole\)](#) im IAM-Benutzerhandbuch. Wenn Sie bereits einen IAM-Benutzer erstellt haben, fahren Sie mit dem nächsten Schritt fort.
 - b. Hängen Sie die Berechtigungen an Ihren IAM-Benutzer an:

- i. Wählen Sie auf der Seite Berechtigungen festlegen die Option Bestehende Richtlinien direkt anhängen aus.
- ii. Suchen Sie nach der IDTUsageIAMPermissionsMetrik-Richtlinie, die Sie im vorherigen Schritt erstellt haben. Markieren Sie das Kontrollkästchen.
- c. Wählen Sie Weiter: Tags aus.
- d. Wählen Sie Next: Review (Weiter: Überprüfen), um eine Zusammenfassung Ihrer Auswahlmöglichkeiten anzuzeigen.
- e. Wählen Sie Create user (Benutzer erstellen) aus.
- f. Um die Zugriffsschlüssel des Benutzers (Zugriffsschlüssel IDs und geheime Zugriffsschlüssel) anzuzeigen, wählen Sie neben dem Passwort und dem Zugriffsschlüssel die Option Anzeigen aus. Zum Speichern der Zugriffsschlüssel wählen Sie Download .csv (CSV-Datei herunterladen) aus und speichern die Datei an einem sicheren Speicherort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.

So konfigurieren Sie Berechtigungen für IDT (AWS CLI)

Gehen Sie wie folgt vor, AWS CLI um Berechtigungen für IDT für AWS IoT Greengrass zu konfigurieren. Wenn Sie bereits Berechtigungen in der Konsole konfiguriert haben, fahren Sie mit [the section called “Konfigurieren Sie Ihr Gerät für die Ausführung von IDT-Tests”](#) oder [the section called “Optional: Konfigurieren des Docker-Containers”](#) fort.

1. Installieren und konfigurieren Sie das auf Ihrem Computer, AWS CLI falls es noch nicht installiert ist. Folgen Sie den Schritten [unter Installation von AWS CLI](#) im AWS Command Line Interface Benutzerhandbuch.

 Note

Das AWS CLI ist ein Open-Source-Tool, mit dem Sie über Ihre AWS Befehlszeilen-Shell mit Diensten interagieren können.

2. Erstellen Sie die folgende vom Kunden verwaltete Richtlinie, die Berechtigungen zur Verwaltung von IDT und Rollen gewährt. AWS IoT Greengrass

Linux, macOS, 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

Dieser Schritt beinhaltet ein Beispiel für eine Windows-Eingabeaufforderung, da er eine andere JSON-Syntax als Linux-, macOS- oder Unix-Terminalbefehle verwendet.

3. Erstellen Sie einen IAM-Benutzer und fügen Sie die von IDT erforderlichen Berechtigungen für hinzu. AWS IoT Greengrass

- a. Erstellen Sie einen IAM-Benutzer.

```
aws iam create-user --user-name user-name
```

- b. Hängen Sie die von Ihnen erstellte IDTUsageMetricsIAMPermissions Richtlinie an Ihren IAM-Benutzer an. *user-name* Ersetzen Sie es durch Ihren IAM-Benutzernamen und *<account-id>* im Befehl durch die ID Ihres AWS-Konto


```
aws iam attach-user-policy --user-name user-name --policy-arn  
arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. Erstellen Sie einen geheimen Zugriffsschlüssel für den Benutzer.

```
aws iam create-access-key --user-name user-name
```

Speichern Sie die Ausgabe an einem sicheren Ort. Sie verwenden diese Informationen später, um Ihre AWS Anmeldeinformationsdatei zu konfigurieren.

Geben Sie die AWS Anmeldeinformationen für IDT ein

Gehen Sie wie folgt vor, damit IDT auf Ihre AWS Anmeldeinformationen zugreifen und Messwerte an AWS senden kann:

1. Speichern Sie die AWS Anmeldeinformationen für Ihren IAM-Benutzer als Umgebungsvariablen oder in einer Anmeldeinformationsdatei:
 - a. Führen Sie den folgenden Befehl aus, um Umgebungsvariablen zu verwenden:

```
AWS_ACCESS_KEY_ID=access-key  
AWS_SECRET_ACCESS_KEY=secret-access-key
```

- b. Um die Datei mit den Anmeldeinformationen zu verwenden, fügen Sie der Datei die folgenden Informationen hinzu `.aws/credentials` file:

```
[profile-name]  
aws_access_key_id=access-key  
aws_secret_access_key=secret-access-key
```

2. Konfigurieren Sie den `auth` Abschnitt der `config.json` Datei. Weitere Informationen finden Sie unter [\(Optional\) Konfigurieren Sie config.json](#).

IDT zur Fehlerbehebung AWS IoT Greengrass

IDT for AWS IoT Greengrass schreibt diese Fehler je nach Art der Fehler an verschiedene Speicherorte. Fehler werden in die Konsole, in Protokolldateien und Testberichte geschrieben.

Fehlercodes

In der folgenden Tabelle finden Sie die von IDT für AWS IoT Greengrass generierten Fehlercodes.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
101	InternalError	Es ist ein interner Fehler aufgetreten.	Überprüfen Sie die Protokolle im Verzeichnis <code><device-tester-extract-location> /results</code> . Wenn Sie das Problem nicht debuggen können, wenden Sie sich an den AWS Entwickler support .
102	TimeoutError	Der Test kann in einem begrenzten Zeitraum nicht abgeschlossen werden. Dies kann in folgenden Fällen vorkommen: Es besteht eine langsame Netzwerkverbindung zwischen Testgerät und Gerät (z. B. bei Verwendung eines VPN-Netzwerks).	- Überprüfen Sie die Netzwerkverbindung und -geschwindigkeit. - Stellen Sie sicher, dass Sie keine Datei im Verzeichnis <code>/test</code> geändert haben. - Versuchen Sie, die fehlerhafte Testgruppe manuell mit dem Flag <code>--</code>

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
		• Ein langsames Netzwerk beeinträchtigt die Kommunikation zwischen Gerät und Cloud. • Das Feld <code>timeout</code> in Testkonfigurationsdateien (<code>test.json</code>) wurde versehentlich geändert.	<code>group-id</code> auszuführen. • Versuchen Sie, die Testsuite auszuführen, indem Sie die Timeouts für die Tests erhöhen. Weitere Informationen finden Sie unter Timeout-Fehler.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
103	PlatformNotSupport Error	Eine falsche Kombination aus Betriebssystem und Architektur ist in <code>device.json</code> angegeben.	Ändern Sie Ihre Konfiguration in eine der unterstützten Kombinationen: • Linux, x86_64 • Linux, ARMv6l • Linux, ARMv7l • Linux, AArch64 • Ubuntu, x86_64 • OpenWRT, ARMv7l • OpenWRT, AArch64 Weitere Informationen finden Sie unter Konfigurieren von device.json.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
104	VersionNotSupportError	Die AWS IoT Greengrass Core-Softwareversion wird von der Version von IDT, die Sie verwenden, nicht unterstützt.	Verwenden Sie den <code>device_tester_bin</code> version Befehl, um die unterstützte Version der AWS IoT Greengrass Core-Software zu finden. Wenn Sie beispielsweise macOS nutzen, verwenden Sie: <code>./devicetester_mac_x86_64</code> version. So finden Sie die Version der AWS IoT Greengrass Core-Software, die Sie verwenden: • Wenn Sie Tests mit vorinstallierter AWS IoT Greengrass Core-Software ausführen, stellen Sie über SSH eine Verbindung zu Ihrem AWS IoT Greengrass Core-Gerät her und starten Sie <code><path-to-preinstalled-green-grass-loc</code>

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			action> /greengrass/ggc/core/greengrassd --version Wenn Sie Tests mit einer anderen Version der AWS IoT Greengrass Core-Software ausführen, wechseln Sie in das devicetester_greengrass_<os>/products/greengrass/gcc Verzeichnis. Die AWS IoT Greengrass Core-Softwareversion ist Teil des ZIP-Dateinamens. Sie können eine andere Version der AWS IoT Greengrass Core-Software testen. Weitere Informationen finden Sie unter Erste Schritte mit AWS IoT Greengrass.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
105	LanguageNotSupportError	IDT unterstützt Python SDKs nur für AWS IoT Greengrass Bibliotheken.	Stellen Sie Folgendes sicher: • Beim SDK-Paket unter <code>devicetester_green_grass_<os>/products/greengrass/ggsdk</code> handelt es sich um das Python SDK. • Der Inhalt des Ordners <code>bin</code> unter <code>devicetester_green_grass_<os>/tests/GGQ_1.0.0/suite/resources/run.runtimefarm/bin</code> wurde nicht geändert.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
106	ValidationError	Manche Felder in <code>device.json</code> oder <code>config.json</code> sind ungültig.	Überprüfen Sie die Fehlermeldung auf der rechten Seite des Fehlercodes im Bericht. • Ungültiger Authentifizierungstyp für Gerät: Geben Sie die richtige Methode zum Herstellen der Verbindung mit Ihrem Gerät an. Weitere Informationen finden Sie unter the section called “Konfigurieren von device.json”. • Ungültiger Pfad zum privaten Schlüssel: Geben Sie den richtigen Pfad zu Ihrem privaten Schlüssel an. Weitere Informationen finden Sie unter Konfigurieren von device.json. •

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			Ungültig AWS-Region: Geben Sie AWS-Region in Ihrer <code>config.js</code> on Datei einen gültigen Wert an. Weitere Informationen finden Sie unter AWS - Service-Endpunkte. AWS Anmeldeinformationen: Legen Sie gültige AWS Anmeldeinformationen auf Ihrem Testcomputer fest (mithilfe von Umgebungsvariablen oder der <code>credentials</code> Datei). Überprüfen Sie, ob das Feld <code>auth</code> korrekt konfiguriert ist. Weitere Informationen finden Sie unter the section called “Erstellen und konfigurieren Sie ein AWS-Konto”.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			Ungültige HSM-Eingabe: Überprüfen Sie die Felder <code>p11Provider</code> , <code>privateKeyLabel</code> , <code>slotLabel</code> , <code>slotUserPin</code> und <code>openSSLEngine</code> in <code>device.json</code> .
107	SSHConnectionFehlgeschlagen	Das Testgerät kann keine Verbindung mit dem konfigurierten Gerät herstellen.	Überprüfen Sie, ob die folgenden Felder in Ihrer Datei <code>device.json</code> korrekt sind: • <code>ip</code> • <code>user</code> • <code>privKeyPath</code> • <code>password</code> Weitere Informationen finden Sie unter Konfigurieren von device.json.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
108	RunCommandError	In einem Test konnte ein Befehl auf dem getesteten Gerät nicht ausgeführt werden.	Überprüfen Sie, ob Root-Zugriff für den konfigurierten Benutzer in <code>device.json</code> zulässig ist. Einige Geräte erfordern ein Passwort, wenn Befehle mit Root-Zugriff ausgeführt werden. Stellen Sie sicher, dass Root-Zugriff ohne Passwort zulässig ist. Weitere Informationen finden Sie in der Dokumentation zu Ihrem Gerät. Versuchen Sie, den fehlgeschlagenen Befehl manuell auf Ihrem Gerät auszuführen, und überprüfen Sie, ob ein Fehler auftritt.
109	PermissionDeniedError	Kein Root-Zugriff.	Richten Sie Root-Zugriff für den konfigurierten Benutzer auf Ihrem Gerät ein.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
110	CreateFileError	Datei kann nicht erstellt werden.	Überprüfen Sie die Speicherplatz- und Verzeichnisberechtigungen Ihres Geräts.
111	CreateDirError	Verzeichnisses kann nicht erstellt werden.	Überprüfen Sie die Speicherplatz- und Verzeichnisberechtigungen Ihres Geräts.
112	InvalidPathError	Der Pfad zur AWS IoT Greengrass Core-Software ist falsch.	Überprüfen Sie, ob der Pfad in der Fehlermeldung gültig ist. Bearbeiten Sie keine Dateien im Verzeichnis <code>devicetester_greengrass_<os></code> .
113	InvalidFileError	Eine Datei ist ungültig.	Überprüfen Sie, ob die Datei in der Fehlermeldung gültig ist.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
114	ReadFileError	Die angegebene Datei kann nicht gelesen werden.	Überprüfen Sie Folgendes: • Dateiberechtigungen sind korrekt. • <code>limits.config</code> erlaubt das Öffnen von genügend Dateien. • Die angegebene Datei in der Fehlermeldung ist vorhanden und gültig. Wenn Sie auf einem macOS testen, erhöhen Sie das Limit für geöffnete Dateien. Das Standardlimit beträgt 256. Dies ist ausreichend für Tests.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
115	FileNotFoundException	Eine erforderliche Datei wurde nicht gefunden.	Überprüfen Sie Folgendes: • Es gibt eine komprimierte Greengrass-Datei unter <code>devicetester_green_grass_ <os>/products/greengrass/ggc</code>. Sie können die AWS IoT Greengrass Core-TAR-Datei von der Downloadseite der AWS IoT Greengrass Kernsoftware herunterladen. • Das SDK-Paket ist unter <code>devicetester_green_grass_ <os>/products/greengrass/gg-sdk</code> vorhanden. • Die Dateien unter <code>devicetester_green_grass_ <os>/</code>

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			tests wurden nicht geändert.
116	OpenFileFailed	Die angegebene Datei konnte nicht geöffnet werden.	Überprüfen Sie Folgendes: - Die angegebene Datei in der Fehlermeldung ist vorhanden und gültig. <code>limits.config</code> erlaubt das Öffnen von genügend Dateien. Wenn Sie auf einem macOS testen, erhöhen Sie das Limit für geöffnete Dateien. Das Standardlimit beträgt 256. Dies ist ausreichend für Tests.
117	WriteFileFailed	Datei konnte nicht geschrieben werden (DUT oder Testsystem).	Stellen Sie sicher, dass das in der Fehlermeldung angegebene Verzeichnis existiert und dass Sie über Schreibberechtigungen verfügen.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
118	FileCleanUpError	Ein Test konnte die angegebene Datei oder das angegebene Verzeichnis nicht entfernen oder der Befehl „umount“ wurde für die angegebene Datei auf dem Remote-Gerät nicht ausgeführt.	Wenn die Binärdatei noch ausgeführt wird, ist die Datei möglicherweise gesperrt. Beenden Sie den Prozess und löschen Sie die angegebene Datei.
119	InvalidInputError	Ungültige Konfiguration.	Überprüfen Sie, ob Ihre Datei <code>suite.json</code> gültig ist.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
120	InvalidCredentialError	Ungültige AWS Anmeldeinformationen.	• Überprüfen Sie Ihre AWS Anmeldeinformationen. Weitere Informationen finden Sie unter the section called “Konfigurieren Sie Ihre AWS Anmeldeinformationen”. • Überprüfen Sie Ihre Netzwerkverbindung und führen Sie die Testgruppe erneut aus. Netzwerkprobleme können diesen Fehler ebenfalls hervorrufen.
121	AWSSessionFehler	Es konnte keine AWS Sitzung erstellt werden.	Dieser Fehler kann auftreten, wenn die AWS Anmeldeinformationen ungültig sind oder die Internetverbindung instabil ist. Versuchen Sie, mit AWS CLI eine AWS API-Operation aufzurufen.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
122	AWSApiCallError	Ein AWS API-Fehler ist aufgetreten.	Der Grund dafür kann ein Netzwerkfehler sein. Überprüfen Sie Ihr Netzwerk, bevor Sie die Testgruppe erneut ausführen.
123	IpNotExistError	IP-Adresse ist nicht in Verbindungsinformationen enthalten.	Überprüfen Sie die Internetverbindung. Sie können die AWS IoT Greengrass Konsole verwenden, um die Verbindungsinformationen für das AWS IoT Greengrass Kernelement zu überprüfen, das für den Test verwendet wird. Wenn in den Verbindungsinformationen 10 Endpunkte enthalten sind, können sie einige oder alle entfernen und den Test erneut ausführen. Weitere Informationen finden Sie unter Verbindungsinformationen .

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
124	OTAJobNotCompleteError	Ein OTA-Auftrag wurde nicht abgeschlossen.	Überprüfen Sie Ihre Netzwerkverbindung und führen Sie die OTA-Testgruppe erneut aus.
125	CreateGreengrassServiceRoleError	Einer der folgenden Punkte ist aufgetreten: • Beim Erstellen einer Rolle ist ein Fehler aufgetreten. • Beim Anhängen einer Richtlinie an die AWS IoT Greengrass Servicerolle ist ein Fehler aufgetreten. • Die mit der Servicerolle verknüpfte Richtlinie ist ungültig. • Beim Zuordnen einer Rolle zu einem ist ein Fehler aufgetreten. AWS-Konto	Konfigurieren Sie die AWS IoT Greengrass Servicerolle. Weitere Informationen finden Sie unter the section called "Greengrass-Servicerolle" .

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
126	DependenciesNotPresentError	Eine oder mehrere für den spezifischen Test erforderliche Abhängigkeiten sind auf dem Gerät nicht vorhanden.	Überprüfen Sie das Testprotokoll, um zu sehen, welche Abhängigkeiten auf Ihrem Gerät fehlen: <i><device-tester-extract-location> /results/<execution-id>/logs/<test-case-name.log></i>
127	Ungültig HSMConfiguration	Die bereitgestellte HSM/PKCS-Konfiguration ist nicht korrekt.	Geben Sie in Ihrer Datei <code>device.json</code> die erforderliche Konfiguration an, um mithilfe von PKCS#11 mit dem HSM zu interagieren.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
128	OTAJobNotSucceededError	Der OTA-Auftrag war nicht erfolgreich.	• Wenn Sie die ota-Testgruppe einzeln ausgeführt haben, führen Sie die ggcdependencies -Testgruppe aus, um sicherzustellen, dass alle Abhängigkeiten (z. B. wget) vorhanden sind. Versuchen Sie dann erneut, die ota-Testgruppe auszuführen. • Rufen Sie die detaillierten Protokolle unter <code><device-tester-extract-location> / results/ <execution-id>/logs/</code> auf, um Informationen zur Fehlerbehebung und zum Fehler zu erhalten. Sehen Sie sich insbesondere die folgenden Protokolle an:

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			• Konsolenprotokoll (test_manager.log) • Protokoll für den OTA-Testfall (ota_test.log) • Protokoll für den GGC-Daemon (ota_test_ggc_logs.tar.gz) • OTA-Agent-Protokoll (ota_test_ota_logs.tar.gz) • Überprüfen Sie Ihre Internetverbindung und führen Sie die ota-Testgruppe erneut aus. • Wenn das Problem weiterhin besteht, wenden Sie sich an den AWS Entwickler-support.

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
129	NoConnectivityError	Der Host-Agent kann keine Verbindung zum Internet herstellen.	Überprüfen Sie Ihre Netzwerkverbindung und die Firewall-Einstellungen. Wiederholen Sie die Testgruppe, nachdem das Verbindungsproblem behoben wurde.
130	NoPermissionError	Der IAM-Benutzer, für den Sie IDT ausführen, ist AWS IoT Greengrass nicht berechtigt, die für die Ausführung von IDT erforderlichen AWS Ressourcen zu erstellen.	Die Richtlinienvorlage , die die für die Ausführung von IDT erforderlichen Berechtigungen gewährt, finden Sie unter Vorlage für Berechtigungsrichtlinien. AWS IoT Greengrass

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
131	LeftoverAgentExist Error	Auf Ihrem Gerät werden AWS IoT Greengrass Prozesse ausgeführt, wenn Sie versuchen, IDT for zu starten. AWS IoT Greengrass	Stellen Sie sicher, dass kein Greengrass-Daemon auf Ihrem Gerät ausgeführt wird. • Sie können den folgenden Befehl verwenden, um einen Daemon zu beenden: <code>sudo ./<absolute-path-to-greengrass-daemon> /greengrassd stop.</code> • Sie können den Greengrass-Daemon auch über PID beenden.  Note Wenn Sie eine bestehende Installation verwenden, die AWS IoT Greengrass so konfiguriert ist, dass sie nach dem

Fehlercode	Name des Fehlercodes	Mögliche Ursache	Fehlerbehebung
			Neustart automatisch gestartet wird, müssen Sie den Daemon nach dem Neustart und vor dem Ausführen der Testsuite beenden.
132	DeviceTimeOffsetError	Das Gerät zeigt die falsche Zeit an.	Stellen Sie Ihr Gerät auf die richtige Zeit ein.
133	Ungültig MLConfiguration	Die angegebene ML-Konfiguration ist falsch.	Geben Sie in Ihrer Datei <code>device.json</code> die richtige Konfiguration an, die zum Ausführen von ML-Inferenztests erforderlich ist. Weitere Informationen finden Sie unter the section called “Optional: Konfigurieren des Geräts für die ML-Qualifizierung” .

IDT auf Fehler beheben AWS IoT Greengrass

Wenn Sie IDT verwenden, müssen Sie die richtigen Konfigurationsdateien einrichten, bevor Sie IDT for ausführen. AWS IoT Greengrass Wenn Sie Parsing- und Konfigurationsfehler erhalten, müssen Sie als Erstes eine für Ihre Umgebung geeignete Konfigurationsvorlage suchen und anwenden.

Wenn weiterhin Probleme auftreten, beachten Sie den folgenden Debugging-Vorgang.

Themen

- [Wo suche ich nach Fehlern?](#)
- [Parsing-Fehler](#)
- [Fehler aufgrund fehlender erforderlicher Parameter](#)
- [Fehler aufgrund eines nicht startenden Tests](#)
- [Fehler wegen fehlender Autorisierung zum Zugriff auf eine Ressource](#)
- [Fehler bei abgelehnter Berechtigung](#)
- [SSH-Verbindungsfehler](#)
- [Timeout-Fehler](#)
- [Fehler bei Tests aufgrund eines fehlenden Befehls](#)
- [Sicherheitsausnahme auf macOS](#)

Wo suche ich nach Fehlern?

High-Level-Fehler werden während der Ausführung auf der Konsole angezeigt. Wenn alle Tests abgeschlossen sind, wird eine Zusammenfassung der fehlgeschlagenen Tests mit dem Fehler angezeigt. `awsiotdevicetester_report.xml` enthält eine Übersicht über alle Fehler, die zum Fehlschlagen eines Tests geführt haben. Die Protokolldateien für die einzelnen Testläufe werden in einem Verzeichnis mit einer UUID als Namen für die Testausführung gespeichert, die während des Testlaufs auf der Konsole angezeigt wurde.

Das Verzeichnis mit den Testprotokollen befindet sich in `<device-tester-extract-location>/results/<execution-id>/logs/`. Dieses Verzeichnis enthält die folgenden Dateien, die für zu Debugging-Zwecken nützlich sind.

Datei	Beschreibung
<code>test_manager.log</code>	

Datei	Beschreibung
	Enthält alle Protokolle, die während der Testausführung zur Konsole geschrieben wurden. Eine Übersicht der Ergebnisse finden Sie am Ende dieser Datei. Diese enthält die Liste der fehlgeschlagenen Tests. Die Warn- und Fehlerprotokolle in dieser Datei können Ihnen Informationen über den/die Fehler bereitstellen.
<code><test-group-id> __<test-name> .log</code>	Detaillierte Protokolle für den jeweiligen Test.
<code><test-name> _ggc_logs.tar.gz</code>	Eine komprimierte Sammlung aller Protokolle, die der AWS IoT Greengrass Core-Daemon während des Tests generiert hat. Weitere Informationen finden Sie unter Fehlerbehebung AWS IoT Greengrass .
<code><test-name> _ota_logs.tar.gz</code>	Eine komprimierte Sammlung von Protokollen, die vom AWS IoT Greengrass OTA-Agenten während des Tests generiert wurden. Nur für OTA-Tests.
<code><test-name> _basic_assertion_publisher_ggad_logs.tar.gz</code>	Eine komprimierte Sammlung von Protokollen, die vom AWS IoT -Herausgebergerät während des Tests erstellt wurden.
<code><test-name> _basic_assertion_subscriber_ggad_logs.tar.gz</code>	Eine komprimierte Sammlung von Protokollen, die vom AWS IoT -Abonnentengerät während des Tests erstellt wurden.

Parsing-Fehler

Es kann vorkommen, dass ein Tippfehler in einer JSON-Konfiguration zu Parsing-Fehlern führt. In den meisten Fällen sind die Ursache des Problems ausgelassene Klammern, Kommas oder Anführungszeichen in Ihrer JSON-Datei. IDT führt eine JSON-Validierung durch und druckt

Debugging-Informationen. Gedruckt werden die Zeile, in der der Fehler aufgetreten ist, sowie Zeilennummer und Spaltennummer des Syntaxfehlers. Diese Informationen sollten ausreichen, um Ihnen bei der Behebung des Fehlers zu helfen. Wenn Sie den Fehler jedoch immer noch nicht finden können, können Sie die Validierung manuell in Ihrer IDE, einem Texteditor wie Atom oder Sublime oder über ein Online-Tool wie JSONLint durchführen.

Fehler aufgrund fehlender erforderlicher Parameter

Da dem IDT neue Funktionen hinzugefügt werden, kann es zu Änderungen an den Konfigurationsdateien kommen. Bei Verwendung einer alten Konfigurationsdatei kann Ihre Konfiguration beschädigt werden. In diesem Fall listet die Datei `<test_case_id>.log` unter `/results/<execution-id>/logs` ausdrücklich alle fehlenden Parameter auf. IDT überprüft zudem Ihre JSON-Konfigurationsdateischemata, um sicherzustellen, dass Sie die neueste unterstützte Version verwenden.

Fehler aufgrund eines nicht startenden Tests

Möglicherweise finden Sie Hinweise auf Fehler beim Teststart. Es gibt mehrere mögliche Ursachen. Gehen Sie daher wie folgt vor:

- Stellen Sie sicher, dass der in Ihrem Ausführungsbefehl enthaltene Poolname tatsächlich vorhanden ist. Auf den Poolnamen wird direkt über Ihre `device.json`-Datei verwiesen.
- Stellen Sie sicher, dass die Geräte in Ihrem Pool über die richtigen Konfigurationsparameter verfügen.

Fehler wegen fehlender Autorisierung zum Zugriff auf eine Ressource

Möglicherweise wird die `<user or role> is not authorized to access this resource`-Fehlermeldung in der Terminalausgabe oder in der `test_manager.log`-Datei unter `/results/<execution-id>/logs` angezeigt. Um dieses Problem zu beheben, fügen Sie die `AWSIoTDeviceTesterForGreengrassFullAccess`-verwaltete Richtlinie an Ihren Testbenutzer an. Weitere Informationen finden Sie unter [the section called “Erstellen und konfigurieren Sie ein AWS-Konto”](#).

Fehler bei abgelehnter Berechtigung

IDT führt Operationen für verschiedene Verzeichnisse und Dateien auf einem zu testenden Gerät aus. Einige dieser Operationen erfordern Stammzugriff. Zum Automatisieren dieser Operationen muss IDT Befehle mit `sudo` ausführen können, ohne ein Passwort einzugeben.

Führen Sie die folgenden Schritte aus, um Sudo-Zugriff zu erteilen, ohne ein Passwort eingeben zu müssen.

 Note

`user` und `username` beziehen sich auf den SSH-Benutzer, der von IDT für den Zugriff auf das zu testende Gerät verwendet wird.

1. Verwenden Sie `sudo usermod -aG sudo <ssh-username>`, um Ihren SSH-Benutzer zur sudo-Gruppe hinzuzufügen.
2. Melden Sie sich ab und melden Sie sich dann wieder an, damit die Änderungen wirksam werden.
3. Öffnen Sie die Datei `/etc/sudoers` und fügen Sie am Ende der Datei die folgende Zeile hinzu:
`<ssh-username> ALL=(ALL) NOPASSWD: ALL`

 Note

Als bewährte Methode empfehlen wir, dass Sie, `sudo visudo` verwenden, wenn Sie `/etc/sudoers` bearbeiten.

SSH-Verbindungsfehler

Wenn IDT keine Verbindung mit einem Gerät herstellen kann, das getestet wird, werden in `/results/<execution-id>/logs/<test-case-id>.log` Verbindungsfehler protokolliert. SSH-Fehlermeldungen befindet sich am Anfang dieser Protokolldatei, da IDT als eine der ersten Operationen eine Verbindung mit einem Gerät herstellt.

Die meisten Windows-Setups verwenden die TTY Pu-Terminal-Anwendung, um eine Verbindung zu Linux-Hosts herzustellen. Diese Anwendung erfordert, dass private PEM-Standardschlüsseldateien in ein proprietäres Windows-Format mit dem Namen PPK konvertiert werden. Wenn IDT in Ihrer `device.json`-Datei konfiguriert ist, verwenden Sie nur PEM-Dateien. Wenn Sie eine PPK-Datei verwenden, kann IDT keine SSH-Verbindung mit dem AWS IoT Greengrass Gerät herstellen und keine Tests ausführen.

Timeout-Fehler

Sie können den Timeout für die einzelnen Tests erhöhen, indem Sie einen Timeout-Multiplikator angeben, der auf den Standardwert des Timeout der einzelnen Tests angewendet wird. Jeder Wert für dieses Kennzeichen muss größer als oder gleich 1,0 sein.

Um den Timeout-Multiplikator zu verwenden, verwenden Sie beim Ausführen des Tests das Flag `--timeout-multiplier`. Zum Beispiel:

```
./devicetester_linux run-suite --suite-id GGQ_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

Führen Sie `run-suite --help` aus, um weitere Informationen zu erhalten.

Fehler bei Tests aufgrund eines fehlenden Befehls

Sie benötigen eine ältere Version der OpenSSL-Bibliothek (libssl1.0.0), um Tests auf Geräten durchzuführen. AWS IoT Greengrass Die meisten gängigen Linux-Verteilungen verwenden libssl Version 1.0.2 oder höher (v1.1.0).

Beispiel: Führen Sie auf einem Raspberry Pi die folgenden Befehle aus, um die erforderliche Version von libssl zu installieren:

1.

```
wget http://ftp.us.debian.org/debian/pool/main/o/openssl/libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```
2.

```
sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```

Sicherheitsausnahme auf macOS

Wenn Sie IDT auf einem Host-Computer ausführen, der macOS 10.15 verwendet, wird das Notarisierungsticket für IDT nicht korrekt erkannt und IDT kann nicht ausgeführt werden. Um IDT auszuführen, müssen Sie der ausführbaren Datei eine Sicherheitsausnahme gewähren.
`devicetester_mac_x86-64`

Um der ausführbaren IDT-Datei eine Sicherheitsausnahme zu gewähren

1. Starten Sie die Systemeinstellungen im Apple-Menü.

2. Wählen Sie Sicherheit und Datenschutz und klicken Sie dann auf der Registerkarte Allgemein auf das Schlosssymbol, um Änderungen an den Sicherheitseinstellungen vorzunehmen.
3. Suchen Sie nach der Nachricht "devicetester_mac_x86-64" was blocked from use because it is not from an identified developer. und wählen Sie „Trotzdem zulassen“.
4. Akzeptieren Sie die Sicherheitswarnung.

Wenn Sie Fragen zur IDT-Supportrichtlinie haben, wenden Sie sich an den [AWS Kundensupport](#).

Support-Richtlinie für AWS IoT Device Tester für AWS IoT Greengrass V1

AWS IoT Device Tester (IDT) for AWS IoT Greengrass ist ein herunterladbares Test-Framework, mit dem Sie Ihre AWS IoT Greengrass Geräte für die Aufnahme in den [AWS Partner Gerätecatalog](#) validieren und [qualifizieren](#) können. Wir empfehlen Ihnen, die neueste Version von AWS IoT Greengrass und IDT zu verwenden, um Ihre Geräte zu testen oder zu qualifizieren. Weitere Informationen finden Sie AWS IoT Greengrass V2 im AWS IoT Greengrass Version 2 Entwicklerhandbuch unter [Unterstützte Versionen von IDT für](#).

Sie können auch jede der unterstützten Versionen von AWS IoT Greengrass und IDT verwenden, um Ihre Geräte zu testen oder zu qualifizieren. Sie können zwar weiterhin [nicht unterstützte Versionen von IDT](#) verwenden, diese Versionen erhalten jedoch keine Bugfixes oder Updates.

Important

Ab dem 4. April 2022 generiert AWS IoT Device Tester (IDT) for AWS IoT Greengrass V1 keine signierten Qualifikationsberichte mehr. Sie können neue AWS IoT Greengrass V1 Geräte nicht mehr über das [Gerätequalifizierungsprogramm für die AWS Partner Aufnahme in den AWS Gerätecatalog qualifizieren](#). Sie können Greengrass V1-Geräte zwar nicht qualifizieren, aber Sie können IDT weiterhin verwenden, um Ihre Greengrass V1-Geräte AWS IoT Greengrass V1 zu testen. Wir empfehlen Ihnen, [IDT for AWS IoT Greengrass V2](#) zu verwenden, um Greengrass-Geräte zu qualifizieren und im [AWS Partner Gerätecatalog](#) aufzulisten.

Wenn Sie Fragen zu den Supportrichtlinien haben, wenden Sie sich an den [AWS Kundenservice](#).

Problembehebung AWS IoT Greengrass

Dieser Abschnitt enthält Informationen zur Fehlerbehebung und mögliche Lösungen zur Behebung von Problemen mit AWS IoT Greengrass.

Informationen zu AWS IoT Greengrass Kontingenten (Limits) finden Sie unter [Service Quotas](#) in der Allgemeine Amazon Web Services-Referenz.

AWS IoT Greengrass Kernprobleme

Wenn die AWS IoT Greengrass Core-Software nicht startet, versuchen Sie es mit den folgenden allgemeinen Schritten zur Fehlerbehebung:

- Stellen Sie sicher, dass Sie die Binärdateien installieren, die für Ihre Architektur geeignet sind. Weitere Informationen finden Sie unter [AWS IoT Greengrass Core-Software](#).
- Stellen Sie sicher, dass Ihr Core-Gerät über lokalen Speicherplatz verfügt. Weitere Informationen finden Sie unter [the section called “Fehlerbehebung bei Speicherproblemen”](#).
- Überprüfen Sie `runtime.log` und `crash.log` auf Fehlermeldungen. Weitere Informationen finden Sie unter [the section called “Fehlerbehebung mit Protokollen”](#).

Suchen Sie nach den folgenden Symptomen und Fehlern, um Informationen zur Behebung von Problemen mit einem AWS IoT Greengrass Core zu finden.

Problembereiche

- [Fehler: In der Konfigurationsdatei fehlt das CaPath, CertPath oder KeyPath. Der Greengrass-Daemon-Prozess mit \[pid = <pid>\] ist abgestürzt.](#)
- [Fehler: Fehler beim Analysieren von /<greengrass-root>/config/config.json.](#)
- [Fehler: Beim Generieren der TLS-Konfiguration ist ein Fehler aufgetreten: ErrUnknown URIScheme](#)
- [Fehler: Laufzeit konnte nicht gestartet werden: Es konnten keine Worker gestartet werden: Zeitüberschreitung für den Container-Test.](#)
- [<address>Fehler: Fehler beim Aufrufen PutLogEvents auf lokaler Cloudwatch, LogGroup:/GreengrassSystem/connection_manager, Fehler:: Sendeanforderung ist fehlgeschlagen, verursacht durch RequestError: Post http://<path>/cloudwatch/logs/: dial tcp: getsockopt: Verbindung verweigert, Antwort: {}.](#)

- <account-id><function-name><version><file-name>Fehler: Server konnte nicht erstellt werden, weil: Gruppe konnte nicht geladen werden: chmod/<greengrass-root>/:aws:lambda: ::function::ggc/deployment/lambda/arn: keine solche Datei oder kein <region>solches Verzeichnis.
- Die AWS IoT Greengrass Core-Software startet nicht, nachdem Sie von der Ausführung ohne Containerisierung zur Ausführung in einem Greengrass-Container gewechselt sind.
- Fehler: Spulengröße sollte mindestens 262144 Bytes betragen.
- Fehler: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {„errorString“: „operation timed out“} (Cloud Messaging Fehler: Fehler trat beim Versuch zur Veröffentlichung einer Nachricht auf)
- <version>Fehler: container_linux.go:344: Das Starten des Container-Prozesses verursachte „process_linux.go:424: container init caused“ rootfs_linux.go:64: mount\\“///_ipc.sock: permission denied\\““. greengrass/ggc/socket/greengrass_ipc.sock\\“ to rootfs \\“/greengrass/ggc/packages rootfs/merged\\“ at \\“/greengrass_ipc.sock\\“ caused \\“stat /greengrass/ggc/socket/greengrass
- Fehler: Greengrass-Daemon wird mit folgender PID ausgeführt: <process-id>. Einige Systemkomponenten konnten nicht gestartet werden. Überprüfen Sie die Datei „runtime.log“ auf Fehler.
- Der Geräteschatten wird nicht mit der Cloud synchronisiert.
- FEHLER: TCP-Verbindung kann nicht angenommen werden. accept tcp [::]:8000: accept4: zu viele geöffnete Dateien.
- Fehler: Laufzeit-Ausführungsfehler: Lambda-Container kann nicht gestartet werden. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\“permission denied\\“\\“.
- Warnung: [WARN] - [5] GK Remote: Fehler beim Abrufen der öffentlichen Schlüsseldaten ErrPrincipalNotConfigured: Der private Schlüssel für MqttCertificate ist nicht festgelegt.
- <account-id><role-name><region>Fehler: Beim Versuch, die Rolle arn:aws:iam: :role/ für den Zugriff auf die S3-URL https://-greengrass-updates.s3 zu verwenden, wurde die Berechtigung verweigert. <region><architecture><distribution-version>.amazonaws.com/core/ /greengrass-core- .tar.gz.
- Der AWS IoT Greengrass Core ist für die Verwendung eines Netzwerk-Proxys konfiguriert und Ihre Lambda-Funktion kann keine ausgehenden Verbindungen herstellen.
- Der Core befindet sich in einer unendlichen Verbinden-Trennen-Schleife. Die Datei „runtime.log“ enthält eine fortlaufende Reihe von Verbinden- und Trennen-Einträgen.

- Fehler: [Lambda-Container kann nicht gestartet werden. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \"proc\" to rootfs \"\""](#)
- [\[FEHLER\] -Laufzeit-Ausführungsfehler: Der Lambda-Container konnte nicht gestartet werden. <ggc-path>{"errorString": „Container-Mounts konnten nicht initialisiert werden: Fehler beim Maskieren der Greengrass-Wurzel im oberen Overlay-Verzeichnis: Fehler beim Erstellen des Maskengeräts im Verzeichnis: Datei existiert"}](#)
- [\[FEHLER\] — Die Bereitstellung ist fehlgeschlagen. {"deploymentId": "<deployment-id>„, „errorString“: „Der Container-Testprozess mit der PID ist <pid>fehlgeschlagen: Status des Container-Prozesses: Exit-Status 1"}](#)
- Fehler: [\[FEHLER\] — Laufzeit-Ausführungsfehler: Der Lambda-Container konnte nicht gestartet werden. <ggc-version><ggc-version>{"errorString": „Container-Mounts konnten nicht initialisiert werden: Fehler beim Erstellen von Overlay fs für Container: Mount-Overlay unter////////greengrass/ggc/packages<ggc-version><ggc-version><ggc-version>/dns:/, rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages upperdir=///rootfs/merged \" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages//rootfs/work\": zu viele Ebenen symbolischer Links greengrass/ggc/packages \"} rootfs/upper,workdir=/greengrass/ggc/packages](#)
- Fehler: [\[DEBUG\] – Routen konnten nicht abgerufen werden. Nachricht wird verworfen.](#)
- Fehler: [\[Errno 24\] Zu viele geöffnete Dateien <lambda-function> \[Errno 24\] Zu viele geöffnete Dateien](#)
- Fehler: [Der DS-Server konnte nicht mit der Überwachung von Socket beginnen: listen unix<ggc-path>/ggc/socket/greengrass_ipc.sock: bind: ungültiges Argument](#)
- [\[INFO\] \(Kopierer\) aws.greengrass. StreamManager: stdout. Verursacht durch: com.fasterxml.jackson.databind. JsonMappingException: Instant überschreitet den Mindest- oder Maximalwert des Sofortwerts](#)
- [GPG error: https://dnw9lb6lp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key](#)

Fehler: In der Konfigurationsdatei fehlt das CaPath, CertPath oder KeyPath. Der Greengrass-Daemon-Prozess mit [pid = <pid>] ist abgestürzt.

Lösung: Dieser Fehler wird möglicherweise angezeigt, `crash.log` wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Dies kann der Fall sein, wenn Sie Version 1.6 oder früher ausführen. Führen Sie eine der folgenden Aktionen aus:

- Führen Sie ein Upgrade auf Version 1.7 oder höher durch. Wir empfehlen, immer die neueste Version der AWS IoT Greengrass Core-Software auszuführen. Informationen zum Download finden Sie unter [AWS IoT Greengrass Core-Software](#).
- Verwenden Sie das richtige `config.json` Format für Ihre AWS IoT Greengrass Core-Softwareversion. Weitere Informationen finden Sie unter [the section called “AWS IoT Greengrass Kernkonfigurationsdatei”](#).

 Note

Um herauszufinden, welche Version der AWS IoT Greengrass Core-Software auf dem Core-Gerät installiert ist, führen Sie die folgenden Befehle in Ihrem Geräteterminal aus.

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

Fehler: Fehler beim Analysieren von /<greengrass-root>/config/config.json.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Stellen Sie sicher, dass die [Greengrass-Konfigurationsdatei](#) ein gültiges JSON-Format verwendet.

Öffnen Sie `config.json` (im Verzeichnis /*greengrass-root*/config) und validieren Sie das JSON-Format. Stellen Sie z. B. sicher, dass Kommas korrekt verwendet werden.

Fehler: Beim Generieren der TLS-Konfiguration ist ein Fehler aufgetreten: ErrUnknown URIScheme

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Stellen Sie sicher, dass die Eigenschaften im [Kryptobereich](#) der Greengrass-Konfigurationsdatei gültig sind. Die Fehlermeldung sollte weitere Informationen enthalten.

Öffnen Sie `config.json` (in `/greengrass-root/config`) und überprüfen Sie den Abschnitt `crypto`. Beispielsweise müssen Zertifikat- und Schlüsselpfade das richtige URI-Format verwenden und auf den richtigen Speicherort verweisen.

Fehler: Laufzeit konnte nicht gestartet werden: Es konnten keine Worker gestartet werden: Zeitüberschreitung für den Container-Test.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Legen Sie die Eigenschaft `postStartHealthCheckTimeout` in der [Greengrass-Konfigurationsdatei](#) fest. Diese optionale Eigenschaft konfiguriert die Zeitspanne (in Millisekunden), die der Greengrass-Daemon auf das Ende der Zustandsprüfung nach dem Start wartet. Der Standardwert ist 30 Sekunden (30.000 ms).

Öffnen Sie `config.json` (im Verzeichnis `/greengrass-root/config`). Fügen Sie im Objekt `runtime` die Eigenschaft `postStartHealthCheckTimeout` hinzu und stellen Sie den Wert auf eine Zahl größer als 30000 ein. Fügen Sie gegebenenfalls ein Komma ein, um eine gültige JSON-Datei zu erstellen. Beispiel:

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

<address>Fehler: Fehler beim Aufrufen PutLogEvents auf lokaler Cloudwatch, LogGroup:/GreengrassSystem/connection_manager, Fehler:: Sendeanforderung ist fehlgeschlagen, verursacht durch RequestError: Post http://<path>/cloudwatch/logs/: dial tcp: getsockopt: Verbindung verweigert, Antwort: {}.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Dies kann auftreten, wenn Sie auf einem Raspberry Pi arbeiten und die erforderliche Speichereinrichtung noch nicht abgeschlossen wurde. AWS IoT Greengrass Weitere Informationen finden Sie [in diesem Schritt](#).

<account-id><function-name><version><file-name>Fehler: Server konnte nicht erstellt werden, weil: Gruppe konnte nicht geladen werden: chmod/<greengrass-root>/:aws:lambda: ::function::/ggc/deployment/lambda/arn: keine solche Datei oder kein <region>solches Verzeichnis.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Wenn Sie eine [ausführbare Lambda-Datei](#) für den Core bereitgestellt haben, überprüfen Sie die Handler Eigenschaft der Funktion in der group.json Datei (in/*greengrass-root*/ggc/deployment/group). Wenn der Handler nicht genau dem Namen Ihrer kompilierten ausführbaren Datei entspricht, ersetzen Sie den Inhalt der Datei group.json durch ein leeres JSON-Objekt ({}) und führen Sie die folgenden Befehle zum Starten von AWS IoT Greengrass aus:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

Verwenden Sie dann die [AWS Lambda -API](#), um den handler-Parameter der Funktionskonfiguration zu aktualisieren, eine neue Funktionsversion zu veröffentlichen und den Alias zu aktualisieren. Weitere Informationen finden Sie unter [AWS Lambda -Funktionsversionen und -Aliase](#).

Vorausgesetzt, Sie haben die Funktion Ihrer Greengrass-Gruppe per Alias hinzugefügt (empfohlen), können Sie Ihre Gruppe jetzt erneut bereitstellen. (Ist dies nicht der Fall, müssen Sie auf die neue

Version oder den Alias der Funktion in Ihrer Gruppendefinition und Ihren Abonnements verweisen, bevor Sie die Gruppe bereitstellen).

Die AWS IoT Greengrass Core-Software startet nicht, nachdem Sie von der Ausführung ohne Containerisierung zur Ausführung in einem Greengrass-Container gewechselt sind.

Lösung: Prüfen Sie, ob Container-Abhängigkeiten fehlen.

Fehler: Spulengröße sollte mindestens 262144 Bytes betragen.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Öffnen Sie die Datei `group.json` (im Verzeichnis `/greengrass-root/ggc/deployment/group`), ersetzen Sie den Inhalt der Datei mit einem leeren JSON-Objekt (`{}`) und führen Sie die folgenden Befehle aus, um AWS IoT Greengrass zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

Anschließend befolgen Sie die Schritte im Verfahren [the section called "So speichern Sie Nachrichten im lokalen Speicher"](#). Stellen Sie für die `GGCloudSpooler`-Funktion sicher, dass Sie einen `GG_CONFIG_MAX_SIZE_BYTES`-Wert angeben, der größer als oder gleich 262144.

Fehler: [ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"} (Cloud Messaging Fehler: Fehler trat beim Versuch zur Veröffentlichung einer Nachricht auf)

Lösung: Möglicherweise wird dieser Fehler in `GGCloudSpooler.log` angezeigt, wenn der Greengrass-Core keine MQTT-Nachrichten an AWS IoT Core senden kann. Dies kann auftreten, wenn die Core-Umgebung eine begrenzte Bandbreite und eine hohe Latenz aufweist.

[Wenn Sie AWS IoT Greengrass Version v1.10.2 oder höher ausführen, versuchen Sie, den](#)

[mqttOperationTimeout](#) Wert in der Datei `config.json` zu erhöhen. Wenn die Eigenschaft nicht vorhanden ist, fügen Sie sie dem `coreThing`-Objekt hinzu. Beispiel:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

Der Standardwert ist 5, und der Mindestwert ist 5.

<version>Fehler: container_linux.go:344: Das Starten des Container-Prozesses verursachte „process_linux.go:424: container init caused \" rootfs_linux.go:64: mount\\\\"_ipc.sock: permission denied\\\" \"\". greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat / greengrass/ggc/socket/greengrass

Lösung: Dieser Fehler tritt `runtime.log` möglicherweise auf, AWS IoT Greengrass wenn die Core-Software nicht gestartet wird. Dies tritt auf, wenn Ihr `umask` höher ist als `0022`. Sie müssen `umask` auf `0022` oder niedriger festlegen, um das Problem zu beheben. Ein Wert von `0022` gewährt standardmäßig jeder Person Leseberechtigung für neue Dateien.

Fehler: Greengrass-Daemon wird mit folgender PID ausgeführt: <process-id>. Einige Systemkomponenten konnten nicht gestartet werden. Überprüfen Sie die Datei „`runtime.log`“ auf Fehler.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Überprüfen Sie `runtime.log` und `crash.log` auf spezifische

Fehlerinformationen. Weitere Informationen finden Sie unter [the section called “Fehlerbehebung mit Protokollen”](#).

Der Geräteschatten wird nicht mit der Cloud synchronisiert.

Lösung: Stellen Sie sicher, dass es AWS IoT Greengrass über Berechtigungen `iot:UpdateThingShadow` und `iot:GetThingShadow` Aktionen in der [Greengrass-Servicerolle](#) verfügt. Wenn die Servicerolle die verwaltete Richtlinie `AWSGreengrassResourceAccessRolePolicy` verwendet, sind diese Berechtigungen standardmäßig enthalten.

Siehe [Beheben von Timeout-Problemen während der Schattensynchronisierung](#).

FEHLER: TCP-Verbindung kann nicht angenommen werden. accept tcp [::]:8000: accept4: zu viele geöffnete Dateien.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in der `greengrassd`-Skriptausgabe angezeigt. Dies kann der Fall sein, wenn das Limit für Dateideskriptoren für die AWS IoT Greengrass Core-Software den Schwellenwert erreicht hat und erhöht werden muss.

Verwenden Sie den folgenden Befehl und starten Sie dann die AWS IoT Greengrass Core-Software neu.

```
ulimit -n 2048
```

Note

In diesem Beispiel wird das Limit auf 2048 erhöht. Wählen Sie einen für Ihren Anwendungsfall geeigneten Wert.

Fehler: Laufzeit-Ausführungsfehler: Lambda-Container kann nicht gestartet werden. container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"".

Lösung: Installieren Sie entweder AWS IoT Greengrass direkt im Stammverzeichnis oder stellen Sie sicher, dass das Verzeichnis, in dem die AWS IoT Greengrass Core-Software installiert ist, und die übergeordneten Verzeichnisse über execute Zugriffsrechte für alle Benutzer verfügen.

Warnung: [WARN] - [5] GK Remote: Fehler beim Abrufen der öffentlichen Schlüsseldaten ErrPrincipalNotConfigured: Der private Schlüssel für MqttCertificate ist nicht festgelegt.

Lösung: AWS IoT Greengrass verwendet einen gemeinsamen Handler, um die Eigenschaften aller Sicherheitsprinzipale zu überprüfen. Diese Warnmeldung in runtime.log wird erwartet, es sei denn, Sie haben einen benutzerdefinierten privaten Schlüssel für den lokalen MQTT-Server angegeben. Weitere Informationen finden Sie unter [the section called "Sicherheitsprinzipale"](#).

<account-id><role-name><region>Fehler: Beim Versuch, die Rolle arn:aws:iam: :role/ für den Zugriff auf die S3-URL https://-greengrass-updates.s3 zu verwenden, wurde die Berechtigung verweigert.

<region><architecture><distribution-version>.amazonaws.com/core/ / greengrass-core- .tar.gz.

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn ein (OTA-) Update fehlschlägt. over-the-air Fügen Sie in der Rollenrichtlinie für den Unterzeichner das Ziel AWS-Region als Resource hinzu. Diese Unterzeichnerrolle wird verwendet, um die S3-URL für das Softwareupdate vorab zu signieren. AWS IoT Greengrass Weitere Informationen finden Sie unter [S3-URL-Signer-Rolle](#).

Der AWS IoT Greengrass Core ist für die Verwendung eines [Netzwerk-Proxys](#) konfiguriert und Ihre Lambda-Funktion kann keine ausgehenden Verbindungen herstellen.

Lösung: Abhängig von Ihrer Laufzeit und den ausführbaren Dateien, die von der Lambda-Funktion zum Herstellen von Verbindungen verwendet werden, erhalten Sie möglicherweise auch Verbindungs-Timeout-Fehler. Stellen Sie sicher, dass Ihre Lambda-Funktionen die entsprechende Proxykonfiguration verwenden, um eine Verbindung über den Netzwerk-Proxy herzustellen. AWS IoT Greengrass übergibt die Proxykonfiguration über die `no_proxy` Umgebungsvariablen `http_proxy` und `https_proxy`, und an benutzerdefinierte Lambda-Funktionen. Sie können, wie im folgenden Python-Ausschnitt gezeigt, aufgerufen werden.

```
import os
print(os.environ['http_proxy'])
```

Verwenden Sie die gleiche Groß-/Kleinschreibung wie die Variable, die in Ihrer Umgebung definiert ist, z. B. nur Kleinbuchstaben (`http_proxy`) oder nur Großbuchstaben (`HTTP_PROXY`). AWS IoT Greengrass unterstützt für diese Variablen beide.

 Note

Die meisten gängigen Bibliotheken, die verwendet werden, um Verbindungen (z. B. `boto3` oder `cURL` und `Python-requests`-Pakete) herzustellen, verwenden diese Umgebungsvariablen standardmäßig.

Der Core befindet sich in einer unendlichen Verbinden-Trennen-Schleife. Die Datei „`runtime.log`“ enthält eine fortlaufende Reihe von Verbinden- und Trennen-Einträgen.

Lösung: Dies kann auftreten, wenn ein anderes Gerät für die Verwendung des Core-Dingnamens als Client-ID für MQTT-Verbindungen mit AWS IoT hartkodiert ist. Gleichzeitige Verbindungen im selben AWS-Region und AWS-Konto müssen einen eindeutigen Client verwenden IDs. Standardmäßig verwendet der Kern den Core-Objektnamen als Client-ID für diese Verbindungen.

Zur Behebung dieses Problems können Sie die Client-ID ändern, die vom anderen Gerät für die Verbindung verwendet wird (empfohlen) oder den Standardwert für den Core überschreiben.

So überschreiben Sie die standardmäßige Client-ID für das Core-Gerät

1. Führen Sie den folgenden Befehl aus, um den Greengrass-Daemon zu beenden:

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

2. Öffnen Sie `greengrass-root/config/config.json` zur Bearbeitung als su-Benutzer.
3. Fügen Sie im `coreThing`-Objekt die `coreClientId`-Eigenschaft hinzu und legen Sie für den Wert Ihre benutzerdefinierte Client-ID fest. Der Wert muss zwischen 1 und 128 Zeichen enthalten. Es muss in der aktuellen Version AWS-Region für eindeutig sein. AWS-Konto

```
"coreClientId": "MyCustomClientId"
```

4. Starten Sie den -Daemon.

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

Fehler: Lambda-Container kann nicht gestartet werden.

container_linux.go:259: starting container process caused

"process_linux.go:345: container init caused \"rootfs_linux.go:62: mounting \\\"proc\\\" to rootfs \\\""

Lösung: Auf einigen Plattformen wird dieser Fehler möglicherweise angezeigt, `runtime.log` wenn AWS IoT Greengrass versucht wird, das `/proc` Dateisystem zu mounten, um einen Lambda-Container zu erstellen. Oder Sie sehen möglicherweise ähnliche Fehler, wie z. B. `operation not permitted` oder `EPERM`. Diese Fehler können auch dann auftreten, wenn Tests auf der Plattform durch das Skript des Abhängigkeitenprüfers ausgeführt werden.

Probieren Sie eine der folgenden möglichen Lösungen aus:

- Aktivieren Sie die Option `CONFIG_DEVPTS_MULTIPLE_INSTANCES` im Linux-Kernel.

- Legen Sie die `/proc`-Mountingoptionen auf dem Host nur auf `rw,relatim` fest.
- Aktualisieren Sie den Linux-Kernel auf 4.9 oder höher.

 Note

Dieses Problem steht nicht im Zusammenhang mit dem Mounten von `/proc` für den lokalen Ressourcenzugriff.

[FEHLER] -Laufzeit-Ausführungsfehler: Der Lambda-Container konnte nicht gestartet werden. `<ggc-path>{"errorString": „Container-Mounts konnten nicht initialisiert werden: Fehler beim Maskieren der Greengrass-Wurzel im oberen Overlay-Verzeichnis: Fehler beim Erstellen des Maskengeräts im Verzeichnis: Datei existiert"}`

Lösung: Möglicherweise wird dieser Fehler in `runtime.log` angezeigt, wenn die Bereitstellung fehlschlägt. Dieser Fehler tritt auf, wenn eine Lambda-Funktion in der AWS IoT Greengrass Gruppe nicht auf das `/usr` Verzeichnis im Dateisystem des Kerns zugreifen kann.

Um dieses Problem zu beheben, fügen Sie der Gruppe eine lokale Volume-Ressource hinzu und stellen Sie dann die Gruppe bereit. Diese Ressource muss:

- `/usr` als Quellpfad und Zielpfad angeben
- Automatisch Betriebssystem-Gruppenberechtigungen der Linux-Gruppe hinzufügen, zu der die Ressource gehört
- Seien Sie mit der Lambda-Funktion verbunden und gewähren Sie schreibgeschützten Zugriff.

[FEHLER] — Die Bereitstellung ist fehlgeschlagen. {"deploymentId": "<deployment-id>,, „errorString“: „Der Container-Testprozess mit der PID ist <pid>fehlgeschlagen: Status des Container-Prozesses: Exit-Status 1"}

Lösung: Möglicherweise wird dieser Fehler in `runtime.log` angezeigt, wenn die Bereitstellung fehlschlägt. Dieser Fehler tritt auf, wenn eine Lambda-Funktion in der AWS IoT Greengrass Gruppe nicht auf das `/usr` Verzeichnis im Dateisystem des Kerns zugreifen kann.

Sie können überprüfen, ob dies der Fall ist, indem Sie nach weiteren Fehlern `GGCanary.log` suchen. Wenn die Lambda-Funktion nicht auf das `/usr` Verzeichnis zugreifen kann, `GGCanary.log` wird der folgende Fehler angezeigt:

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or directory"
```

Um dieses Problem zu beheben, fügen Sie der Gruppe eine lokale Volume-Ressource hinzu und stellen Sie dann die Gruppe bereit. Diese Ressource muss:

- `/usr` als Quellpfad und Zielpfad angeben
- Automatisch Betriebssystem-Gruppenberechtigungen der Linux-Gruppe hinzufügen, zu der die Ressource gehört
- Seien Sie mit der Lambda-Funktion verbunden und gewähren Sie schreibgeschützten Zugriff.

Fehler: [FEHLER] — Laufzeit-Ausführungsfehler: Der Lambda-Container konnte nicht gestartet werden. `<ggc-version><ggc-version>{"errorString": „Container-Mounts konnten nicht initialisiert werden: Fehler beim Erstellen von Overlay fs für Container: Mount-Overlay unter////////greengrass/ggc/packages<ggc-version><ggc-version><ggc-version>/dns:/, rootfs/merged failed: failed to mount with args source=\"/no_source\" dest=\"/greengrass/ggc/packages upperdir=///rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"/lowerdir=/greengrass/ggc/packages//rootfs/work\": zu viele Ebenen symbolischer Links greengrass/ggc/packages \"} rootfs/upper,workdir=/greengrass/ggc/packages`

Lösung: Möglicherweise wird dieser Fehler in der Datei angezeigt, wenn die Core-Software nicht gestartet wird. `runtime.log` AWS IoT Greengrass Dieses Problem tritt auf Debian-Betriebssystemen möglicherweise häufiger auf.

Führen Sie folgende Schritte aus, um dieses Problem zu lösen:

1. Aktualisieren Sie die AWS IoT Greengrass Core-Software auf Version 1.9.3 oder höher. Dadurch sollte das Problem automatisch behoben werden.
2. Wenn dieser Fehler nach dem Upgrade der AWS IoT Greengrass Core-Software weiterhin angezeigt wird, setzen Sie die `system.useOverlayWithTmpfs` Eigenschaft `true` in der [Datei config.json](#) auf.

Example Beispiel

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

```
}
```

Note

Ihre AWS IoT Greengrass Core-Softwareversion wird in der Fehlermeldung angezeigt. Führen Sie `uname -r` aus, um Ihre Linux-Kernel-Version zu suchen.

Fehler: [DEBUG] – Routen konnten nicht abgerufen werden. Nachricht wird verworfen.

Lösung: Überprüfen Sie die Abonnements in Ihrer Gruppe und stellen Sie sicher, dass das in der [DEBUG]-Nachricht aufgelistete Abonnement vorhanden ist.

Fehler: [Errno 24] Zu viele geöffnete Dateien <lambda-function> [Errno 24] Zu viele geöffnete Dateien

Lösung: Möglicherweise wird dieser Fehler in Ihrer Lambda-Funktionsprotokolldatei angezeigt, wenn die Funktion `StreamManagerClient` im Funktionshandler instanziiert wird. Es wird empfohlen, den Client außerhalb des Handlers zu erstellen. Weitere Informationen finden Sie unter [the section called “Wird verwendet StreamManagerClient , um mit Streams zu arbeiten”](#).

Fehler: Der DS-Server konnte nicht mit der Überwachung von Socket beginnen: listen unix<ggc-path>/ggc/socket/greengrass_ipc.sock: bind: ungültiges Argument

Lösung: Dieser Fehler wird möglicherweise angezeigt, wenn die AWS IoT Greengrass Core-Software nicht gestartet wird. Dieser Fehler tritt auf, wenn die AWS IoT Greengrass Core-Software in einem Ordner mit einem langen Dateipfad installiert wird. Installieren Sie die AWS IoT Greengrass Core-Software erneut in einem Ordner mit einem Dateipfad, der weniger als 79 Byte umfasst, wenn Sie kein [Schreibverzeichnis](#) verwenden, oder 83 Byte, wenn Sie ein Schreibverzeichnis verwenden.

[INFO] (Kopierer) aws.greengrass.StreamManager: stdout. Verursacht durch: com.fasterxml.jackson.databind.JsonMappingException: Instant überschreitet den Mindest- oder Maximalwert des Sofortwerts

Wenn Sie die AWS IoT Greengrass Kernsoftware auf Version 1.11.3 aktualisieren, wird möglicherweise der folgende Fehler in den Stream Manager-Protokollen angezeigt, wenn der Stream Manager nicht gestartet werden kann.

```
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}
```

Wenn Sie eine Version der AWS IoT Greengrass Kernsoftware verwenden, die älter als v1.11.3 ist, und Sie auf eine neuere Version aktualisieren möchten, verwenden Sie ein OTA-Update, um auf v1.11.4 zu aktualisieren.

GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

Wenn Sie `apt update` auf einem Gerät arbeiten, auf dem Sie [die AWS IoT Greengrass Kernsoftware aus einem APT-Repository installiert haben](#), wird möglicherweise der folgende Fehler angezeigt.

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```


Dieser Fehler tritt auf, weil AWS IoT Greengrass es nicht mehr möglich ist, die AWS IoT Greengrass Kernsoftware aus dem APT-Repository zu installieren oder zu aktualisieren. Um das Repository erfolgreich auszuführen `apt update`, entfernen Sie das AWS IoT Greengrass Repository aus der Quellenliste des Geräts.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

Bereitstellungsprobleme

Im Folgenden finden Sie Informationen zur Behebung von Problemen mit der Bereitstellung.

Problembereiche

- [Ihre aktuelle Bereitstellung funktioniert nicht und Sie möchten zu einer früheren, funktionierenden Bereitstellung zurückkehren.](#)
- [Für die Bereitstellung wird der Fehler „403 Forbidden“ in den Protokollen angezeigt.](#)
- [Ein ConcurrentDeployment Fehler tritt auf, wenn Sie den Befehl create-deployment zum ersten Mal ausführen.](#)
- [Fehler: Greengrass ist nicht zur Übernahme der Servicerolle berechtigt, die mit diesem Konto verknüpft ist; oder der Fehler: Fehlgeschlagen: TES-Servicerolle ist nicht mit diesem Konto verknüpft.](#)
- [Fehler: Download-Schritt in der Bereitstellung kann nicht ausgeführt werden. Fehler beim Herunterladen: Fehler beim Herunterladen der Gruppendefinitionsdatei:... x509: Zertifikat ist abgelaufen oder noch nicht gültig](#)
- [Die Bereitstellung wird nicht abgeschlossen.](#)
- [Fehler: Die ausführbaren Java- oder Java8-Dateien konnten nicht gefunden werden, oder der Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen. Fehler: Der Worker konnte nicht mit einem <worker-id>bestimmten Grund initialisiert werden. Die installierte Java-Version muss größer oder gleich 8 sein](#)
- [Die Bereitstellung wird nicht fertiggestellt und „runtime.log“ enthält mehrere Einträge für „1 Sekunde auf Anhalten des Containers gewartet“.](#)
- [Die Bereitstellung wird nicht abgeschlossen und runtime.log enthält "\[FEHLER\]-Greengrass-Bereitstellungsfehler: Fehler beim Melden des Bereitstellungsstatus an die Cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>,"](#)

error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"

- <path>Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen Fehler: Fehler bei der Verarbeitung. Die Gruppenkonfiguration ist ungültig: 112 oder [119 0] haben keine RW-Rechte für die Datei:.
- Fehler: < list-of-function-arns > sind für die Ausführung als Root konfiguriert, aber Greengrass ist nicht für die Ausführung von Lambda-Funktionen mit Root-Rechten konfiguriert.
- Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen Fehler: Greengrass-Bereitstellungsfehler: Download-Schritt in der Bereitstellung konnte nicht ausgeführt werden. Fehler bei der Verarbeitung: Die heruntergeladene Gruppendatei konnte nicht geladen werden: Die UID konnte nicht anhand des Benutzernamens gefunden werden, userName: ggc_user: user: unknown user ggc_user.
- Fehler: [FEHLER]-Laufzeitausführungsfehler: Lambda-Container kann nicht gestartet werden. {"errorString": "Fehler beim Initialisieren von Container-Mounts: Fehler beim Maskieren des Greengrass-Stamms im Overlay-Oberverzeichnis: Fehler beim Erstellen eines Maskengeräts im Verzeichnis <ggc-path>: Datei vorhanden"}
- Fehler: Bereitstellung <deployment-id>des Typs NewDeployment für Gruppe ist <group-id>fehlgeschlagen Fehler: Prozessstart fehlgeschlagen: container_linux.go:259: Das Starten des Container-Prozesses verursachte „process_linux.go:250: running exec setns process for init caused\" wait: no child processes\" "".
- Fehler: [WARN<host-prefix>] -MQTT [Client] Wählen Sie TCP: Lookup -ats.iot. <region>.amazonaws.com: kein solcher Host... [FEHLER] -Greengrass-Bereitstellungsfehler: Der Bereitstellungsstatus konnte nicht an die Cloud zurückgemeldet werden... net/http: Die Anfrage wurde beim Warten auf die Verbindung abgebrochen (Client.Timeout wurde beim Warten auf Header überschritten)

Ihre aktuelle Bereitstellung funktioniert nicht und Sie möchten zu einer früheren, funktionierenden Bereitstellung zurückkehren.

Lösung: Verwenden Sie die AWS IoT Konsole oder AWS IoT Greengrass API, um eine zuvor funktionierende Bereitstellung erneut bereitzustellen. Hierdurch wird die entsprechende Gruppenversion auf Ihrem Core-Gerät bereitgestellt.

So stellen Sie eine Bereitstellung erneut bereit (Konsole)

1. Wählen Sie auf der Seite mit der Gruppenkonfiguration die Registerkarte Bereitstellungen aus. Diese Seite zeigt den Bereitstellungsverlauf für die Gruppe an, einschließlich Datum und Uhrzeit, Gruppenversion und Status der einzelnen Bereitstellungsversuche.
2. Suchen Sie die Zeile mit der Bereitstellung, die Sie erneut bereitstellen möchten. Wählen Sie die Bereitstellung aus, die Sie erneut bereitstellen möchten, und klicken Sie auf Erneut bereitstellen.

Deployments	Group history overview			By deployment ▼
Subscriptions	Deployed	Version	Status	
Cores				
Devices	Jul 1, 2019 1:56:49 PM -0700	8dd1d899-4ac9-4f5d-afe4-22de086efc62	● Successfully complet...	...
Lambdas	Jul 1, 2019 1:41:47 PM -0700	4ad66e5d-3808-446b-940a-b1a788898382	● Successfully complet...	...
Resources				Re-deploy
Connectors	Jun 18, 2019 8:16:02 AM -0700	1f3870b6-850e-4c97-8018-c872e17b235b	● Failed	...

So stellen Sie eine Bereitstellung erneut bereit (CLI)

1. Verwenden Sie diese [ListDeployments](#) Option, um die ID der Bereitstellung zu ermitteln, die Sie erneut bereitstellen möchten. Beispiel:

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcdf7
```

Der Befehl gibt die Liste der Bereitstellungen für die Gruppe zurück.

```
{
  "Deployments": [
    {
      "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
      "DeploymentType": "NewDeployment",
      "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/74d0b623-c2f2-4cad-9acc-ef92f61fcdf7/versions/8dd1d899-4ac9-4f5d-afe4-22de086efc62",
      "CreatedAt": "2019-07-01T20:56:49.641Z"
    },
    {
      "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
      "DeploymentType": "NewDeployment",

```

```

      "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
      "CreatedAt": "2019-07-01T20:41:47.048Z"
    },
    {
      "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
      "DeploymentType": "NewDeployment",
      "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
      "CreatedAt": "2019-06-18T15:16:02.965Z"
    }
  ]
}

```

Note

Diese AWS CLI Befehle verwenden Beispielwerte für die Gruppe und die Bereitstellung-ID. Wenn Sie die Befehle ausführen, müssen Sie die Beispielwerte ersetzen.

2. Wird verwendet [CreateDeployment](#), um die Zielbereitstellung erneut bereitzustellen. Legen Sie den Bereitstellungstyp auf Redeployment fest. Beispiel:

```

aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596

```

Der Befehl gibt den ARN und die ID der neuen Bereitstellung zurück.

```

{
  "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
  "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
}

```

3. Wird verwendet [GetDeploymentStatus](#), um den Status der Bereitstellung abzurufen.

Für die Bereitstellung wird der Fehler „403 Forbidden“ in den Protokollen angezeigt.

Lösung: Stellen Sie sicher, dass die Richtlinie des AWS IoT Greengrass Kerns in der Cloud eine zulässige Aktion umfasst `"greengrass:*"`.

Ein `ConcurrentDeployment` Fehler tritt auf, wenn Sie den Befehl `create-deployment` zum ersten Mal ausführen.

Lösung: Möglicherweise wird eine Bereitstellung verarbeitet. Sie können [get-deployment-status](#) ausführen, um zu prüfen, ob eine Bereitstellung erstellt wurde. Falls nicht, versuchen Sie erneut, die Bereitstellung zu erstellen.

Fehler: Greengrass ist nicht zur Übernahme der Servicerolle berechtigt, die mit diesem Konto verknüpft ist; oder der Fehler: Fehlgeschlagen: TES-Servicerolle ist nicht mit diesem Konto verknüpft.

Lösung: Möglicherweise wird Ihnen dieser Fehler angezeigt, wenn die Bereitstellung fehlschlägt. Vergewissern Sie sich, dass Ihrer aktuellen AWS-Region Greengrass-Servicerolle eine AWS-Konto Greengrass-Servicerolle zugeordnet ist. Für weitere Informationen siehe [the section called "Verwaltung der Servicerolle \(CLI\)"](#) oder [the section called "Verwaltung der Servicerolle \(Konsole\)"](#).

Fehler: Download-Schritt in der Bereitstellung kann nicht ausgeführt werden. Fehler beim Herunterladen: Fehler beim Herunterladen der Gruppenspezifikationsdatei:... x509: Zertifikat ist abgelaufen oder noch nicht gültig

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in `runtime.log` angezeigt, wenn die Bereitstellung fehlschlägt. Wenn eine `Deployment failed`-Fehlermeldung mit dem Text `x509: certificate has expired or is not yet valid` angezeigt wird, überprüfen Sie die Uhr

des Geräts. TLS- und X.509-Zertifikate bieten eine sichere Grundlage für die Erstellung von IoT-Systemen, erfordern jedoch genaue Zeitangaben auf Servern und Clients. IoT-Geräte sollten die richtige Zeit (innerhalb von 15 Minuten) haben, bevor sie versuchen, eine Verbindung zu AWS IoT Greengrass anderen TLS-Diensten herzustellen, die Serverzertifikate verwenden. Weitere Informationen finden Sie im [AWS offiziellen Blog unter Verwendung von Gerätezeit zur Validierung von AWS IoT Serverzertifikaten](#) im Internet der Dinge.

Die Bereitstellung wird nicht abgeschlossen.

Lösung: Führen Sie die folgenden Schritte aus:

- Stellen Sie sicher, dass der AWS IoT Greengrass Daemon auf Ihrem Core-Gerät läuft. Führen Sie im Terminal Ihres Core-Geräts die folgenden Befehle aus, um zu überprüfen, ob der Daemon läuft, und starten Sie ihn bei Bedarf.

1. So prüfen Sie, ob der Daemon ausgeführt wird:

```
ps aux | grep -E 'greengrass.*daemon'
```

Wenn die Ausgabe einen `root`-Eintrag für `/greengrass/ggc/packages/1.11.6/bin/daemon` enthält, dann wird der Daemon ausgeführt.

Die Version im Pfad hängt von der AWS IoT Greengrass Core-Softwareversion ab, die auf Ihrem Core-Gerät installiert ist.

2. Um den Daemon zu starten:

```
cd /greengrass/ggc/core/  
sudo ./greengrassd start
```

- Stellen Sie sicher, dass Ihr Core-Gerät verbunden ist und die Core-Verbindungsendpunkte ordnungsgemäß konfiguriert sind.

Fehler: Die ausführbaren Java- oder Java8-Dateien konnten nicht gefunden werden, oder der Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen. Fehler: Der Worker konnte nicht mit einem <worker-id>bestimmten Grund initialisiert werden. Die installierte Java-Version muss größer oder gleich 8 sein

Lösung: Wenn Stream Manager für den AWS IoT Greengrass Core aktiviert ist, müssen Sie die Java 8-Runtime auf dem Core-Gerät installieren, bevor Sie die Gruppe bereitstellen. Weitere Informationen finden Sie in den [Anforderungen](#) für den Stream-Manager. Stream Manager ist standardmäßig aktiviert, wenn Sie den Workflow zur Erstellung von Standardgruppen in der AWS IoT Konsole verwenden, um eine Gruppe zu erstellen.

Oder deaktivieren Sie den Stream-Manager und stellen anschließend die Gruppe bereit. Weitere Informationen finden Sie unter [the section called “Konfigurieren der Einstellungen \(Konsole\)”](#).

Die Bereitstellung wird nicht fertiggestellt und „runtime.log“ enthält mehrere Einträge für „1 Sekunde auf Anhalten des Containers gewartet“.

Lösung: Führen Sie die folgenden Befehle in Ihrem Core-Device-Terminal aus, um den AWS IoT Greengrass Daemon neu zu starten.

```
cd /greengrass/ggc/core/  
sudo ./greengrassd stop  
sudo ./greengrassd start
```

Die Bereitstellung wird nicht abgeschlossen und **runtime.log** enthält "[FEHLER]-Greengrass-Bereitstellungsfehler: Fehler beim Melden des Bereitstellungsstatus an die Cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}"

Lösung: Dieser Fehler wird möglicherweise in `runtime.log` angezeigt, wenn der Greengrass Core für die Verwendung einer HTTPS-Proxyverbindung konfiguriert ist und die Zertifikatskette des Proxy-Servers auf dem System nicht vertrauenswürdig ist. Fügen Sie die Zertifikatskette dem CA-Stammzertifikat hinzu, um dieses Problem zu beheben. Der Greengrass Core fügt die Zertifikate aus dieser Datei dem Zertifikatpool hinzu, der für die TLS-Authentifizierung in HTTPS- und MQTT-Verbindungen mit AWS IoT Greengrass verwendet wird.

Das folgende Beispiel zeigt ein CA-Zertifikat des Proxy-Servers, das der Datei mit dem CA-Stammzertifikat hinzugefügt wurde:

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQ0xXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhUL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBjbMUMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/n07f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/1ND/PuMZ/s8Yj1kY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPm1jZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKw
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJioblXgUka642M4UwtBV8oK2xJNDd2ZhwLnQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```


Standardmäßig ist das CA-Stammzertifikat in `/greengrass-root/certs/root.ca.pem` gespeichert. Wenn Sie den Speicherort auf Ihrem Core-Gerät suchen möchten, überprüfen Sie die Eigenschaft `crypto.caPath` in der Datei [config.json](#).

 Note

`greengrass-root` steht für den Pfad, in dem die AWS IoT Greengrass Core-Software auf Ihrem Gerät installiert ist. Normalerweise ist dies das Verzeichnis `/greengrass`.

<path>Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen Fehler: Fehler bei der Verarbeitung. Die Gruppenkonfiguration ist ungültig: 112 oder [119 0] haben keine RW-Rechte für die Datei:.

Lösung: Stellen Sie sicher, dass die Eigentümergruppe des `<path>` Verzeichnisses Lese- und Schreibberechtigungen für das Verzeichnis hat.

Fehler: `< list-of-function-arns >` sind für die Ausführung als Root konfiguriert, aber Greengrass ist nicht für die Ausführung von Lambda-Funktionen mit Root-Rechten konfiguriert.

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in `runtime.log` angezeigt, wenn die Bereitstellung fehlschlägt. Stellen Sie sicher, dass Sie so konfiguriert haben AWS IoT Greengrass , dass Lambda-Funktionen mit Root-Rechten ausgeführt werden können. Ändern Sie entweder den Wert von `allowFunctionsToRunAsRoot` in `greengrass_root/config/config.json` to `yes` oder ändern Sie die Lambda-Funktion so, dass sie unter einem anderen Benutzer/einer anderen Gruppe ausgeführt wird. Weitere Informationen finden Sie unter [the section called “Eine Lambda-Funktion als Root ausführen”](#).

Fehler: Die Bereitstellung <deployment-id>des Typs NewDeployment für die Gruppe ist <group-id>fehlgeschlagen Fehler: Greengrass-Bereitstellungsfehler: Download-Schritt in der Bereitstellung konnte nicht ausgeführt werden. Fehler bei der Verarbeitung: Die heruntergeladene Gruppendatei konnte nicht geladen werden: Die UID konnte nicht anhand des Benutzernamens gefunden werden, userName: ggc_user: user: unknown user ggc_user.

Lösung: Wenn die [Standardzugriffsidentität](#) der AWS IoT Greengrass Gruppe die Standardsystemkonten verwendet, müssen der ggc_user Benutzer und ggc_group die Gruppe auf dem Gerät vorhanden sein. Anweisungen zum Hinzufügen von Benutzer und Gruppe finden Sie in diesem [Schritt](#). Sie müssen die Namen genau wie gezeigt eingeben.

Fehler: [FEHLER]-Laufzeitausführungsfehler: Lambda-Container kann nicht gestartet werden. {"errorString": "Fehler beim Initialisieren von Container-Mounts: Fehler beim Maskieren des Greengrass-Stamms im Overlay-Oberverzeichnis: Fehler beim Erstellen eines Maskengeräts im Verzeichnis <ggc-path>: Datei vorhanden"}

Lösung: Möglicherweise wird Ihnen diese Fehlermeldung in runtime.log angezeigt, wenn die Bereitstellung fehlschlägt. Dieser Fehler tritt auf, wenn eine Lambda-Funktion in der Greengrass-Gruppe nicht auf das /usr Verzeichnis im Dateisystem des Cores zugreifen kann. Um dieses Problem zu beheben, fügen Sie der Gruppe eine [lokale Volume-Ressource](#) hinzu und stellen Sie dann die Gruppe bereit. Die Ressource muss:

- /usr als Quellpfad und Zielpfad angeben
- Automatisch Betriebssystem-Gruppenberechtigungen der Linux-Gruppe hinzufügen, zu der die Ressource gehört
- Seien Sie mit der Lambda-Funktion verbunden und gewähren Sie schreibgeschützten Zugriff.

Fehler: Bereitstellung <deployment-id>des Typs NewDeployment für Gruppe ist <group-id>fehlgeschlagen Fehler: Prozessstart fehlgeschlagen: container_linux.go:259: Das Starten des Container-Prozesses verursachte „process_linux.go:250: running exec setns process for init caused\" wait: no child processes\"“.

Lösung: Möglicherweise wird Ihnen dieser Fehler angezeigt, wenn die Bereitstellung fehlschlägt. Wiederholen Sie die Bereitstellung.

Fehler: [WARN<host-prefix>] -MQTT [Client] Wählen Sie TCP: Lookup -ats.iot. <region>.amazonaws.com: kein solcher Host... [FEHLER] - Greengrass-Bereitstellungsfehler: Der Bereitstellungsstatus konnte nicht an die Cloud zurückgemeldet werden... net/http: Die Anfrage wurde beim Warten auf die Verbindung abgebrochen (Client.Timeout wurde beim Warten auf Header überschritten)

Lösung: Dieser Fehler wird Ihnen möglicherweise angezeigt, wenn Sie systemd-resolved verwenden. Hierdurch wird die Einstellung DNSSEC standardmäßig aktiviert. Daher werden zahlreiche öffentliche Domänen nicht erkannt. Versuche, den AWS IoT Greengrass Endpunkt zu erreichen, können den Host nicht finden, sodass Ihre Bereitstellungen den Status beibehalten. In Progress

Sie können die folgenden Befehle und Ausgaben verwenden, um auf dieses Problem zu testen. Ersetzen Sie den *region* Platzhalter in den Endpunkten durch Ihren. AWS-Region

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
failed-auxiliary
```

Eine mögliche Lösung besteht darin, DNSSEC zu deaktivieren. Wenn DNSSEC false ist, werden DNS-Lookups nicht DNSSEC-validiert. Weitere Informationen finden Sie unter diesem [bekannten Problem](#) für systemd.

1. Fügen Sie `DNSSEC=false` zu `/etc/systemd/resolved.conf` hinzu.
2. Starten Sie `systemd-resolved` neu.

Um Informationen zu `resolved.conf` und DNSSEC zu erhalten, führen Sie man `resolved.conf` in Ihrem Terminal aus.

Probleme beim Erstellen der Gruppe oder Funktion

Verwenden Sie die folgenden Informationen, um Probleme beim Erstellen einer AWS IoT Greengrass Gruppe oder einer Greengrass-Lambda-Funktion zu beheben.

Problembereiche

- [Fehler: Ihre Konfiguration 'IsolationMode' für die Gruppe ist ungültig.](#)
- [Fehler: Ihre Konfiguration IsolationMode " für die Funktion mit arn <function-arn>ist ungültig.](#)
- [Fehler: Die MemorySize Konfiguration für eine Funktion mit arn <function-arn>ist in IsolationMode = nicht erlaubtNoContainer.](#)
- [Fehler: Der Zugriff auf die Sysfs-Konfiguration für eine Funktion mit arn <function-arn>ist in = nicht zulässig. IsolationMode NoContainer](#)
- [Fehler: Die MemorySize Konfiguration für die Funktion mit arn <function-arn>ist in IsolationMode = GreengrassContainer erforderlich.](#)
- [Fehler: Die Funktion <function-arn>bezieht sich auf eine Ressource des Typs<resource-type>, der in IsolationMode = NoContainer nicht zulässig ist.](#)
- [Fehler: Die Ausführungskonfiguration für die Funktion mit dem ARN <function-arn> ist nicht zulässig.](#)

Fehler: Ihre Konfiguration 'IsolationMode' für die Gruppe ist ungültig.

Lösung: Dieser Fehler tritt auf, wenn der Wert `IsolationMode` in der `DefaultConfig` von `function-definition-version` nicht unterstützt wird. Unterstützte Werte sind `GreengrassContainer` und `NoContainer`.

Fehler: Ihre Konfiguration IsolationMode " für die Funktion mit arn <function-arn> ist ungültig.

Lösung: Dieser Fehler tritt auf, wenn der Wert IsolationMode in <function-arn> der function-definition-version nicht unterstützt wird. Unterstützte Werte sind GreengrassContainer und NoContainer.

Fehler: Die MemorySize Konfiguration für eine Funktion mit arn <function-arn> ist in IsolationMode = nicht erlaubtNoContainer.

Lösung: Dieser Fehler tritt auf, wenn Sie einen MemorySize Wert angeben und sich für eine Ausführung ohne Containerisierung entscheiden. Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, dürfen keine Speichergrenzen haben. Sie können entweder das Limit entfernen oder die Lambda-Funktion so ändern, dass sie in einem AWS IoT Greengrass Container ausgeführt wird.

Fehler: Der Zugriff auf die Sysfs-Konfiguration für eine Funktion mit arn <function-arn> ist in IsolationMode NoContainer = nicht zulässig.

Lösung: Dieser Fehler tritt auf, wenn Sie true für AccessSysfs angeben und die Ausführung ohne Containerisierung wählen. Lambda-Funktionen, die ohne Containerisierung ausgeführt werden, muss der Code aktualisiert werden, um direkt auf das Dateisystem zugreifen zu können, und sie können nicht verwendet werden. AccessSysfs Sie können entweder den Wert false für AccessSysfs angeben oder die Lambda-Funktion so ändern, dass sie in einem AWS IoT Greengrass Container ausgeführt wird.

Fehler: Die MemorySize Konfiguration für die Funktion mit arn <function-arn> ist in IsolationMode = GreengrassContainer erforderlich.

Lösung: Dieser Fehler tritt auf, weil Sie kein MemorySize Limit für eine Lambda-Funktion angegeben haben, die Sie in einem AWS IoT Greengrass Container ausführen. Geben Sie einen MemorySize-Wert an, um den Fehler zu beheben.

Fehler: Die Funktion `<function-arn>` bezieht sich auf eine Ressource des Typs `<resource-type>`, der in `IsolationMode = NoContainer` nicht zulässig ist.

Lösung: Sie können nicht auf Ressourcentypen

`Local.DeviceLocal.Volume`, `ML_Model.SageMaker.Job`, `ML_Model.S3_Object`, oder `S3_Object.Generic_Archive` Ressourcentypen zugreifen, wenn Sie eine Lambda-Funktion ohne Containerisierung ausführen. Wenn Sie diese Ressourcentypen benötigen, müssen Sie sie in einem Container ausführen. AWS IoT Greengrass Sie können auch direkt auf lokale Geräte zugreifen, wenn sie ohne Containerisierung ausgeführt werden, indem Sie den Code in Ihrer Lambda-Funktion ändern.

Fehler: Die Ausführungskonfiguration für die Funktion mit dem ARN `<function-arn>` ist nicht zulässig.

Lösung: Dieser Fehler tritt auf, wenn Sie eine Lambda-Systemfunktion mit `GGIPDetector` oder erstellen `GGCloudSpooler` und die `RunAs` Konfiguration `IsolationMode` angegeben haben. Sie müssen die `Execution` Parameter für diese Lambda-Systemfunktion weglassen.

Erkennungsprobleme

Verwenden Sie die folgenden Informationen, um Probleme mit dem AWS IoT Greengrass Discovery-Service zu beheben.

Problembereiche

- [Fehler: Gerät gehört zu vielen Gruppen an; Geräte dürfen nicht mehr als 10 Gruppen angehören](#)

Fehler: Gerät gehört zu vielen Gruppen an; Geräte dürfen nicht mehr als 10 Gruppen angehören

Lösung: Dies ist eine bekannte Einschränkung. Ein [Client-Gerät](#) kann Mitglied von bis zu 10 Gruppen sein.

Probleme mit Machine Learning-Ressourcen

Verwenden Sie die folgenden Informationen, um Probleme mit Machine Learning-Ressourcen zu beheben.

Problembereiche

- [Ungültiger MLModel Besitzer — GroupOwnerSetting wird in der ML-Modellressource angegeben, GroupPermission ist aber GroupOwner nicht vorhanden](#)
- [NoContainer Die Funktion kann beim Anhängen von Ressourcen für Machine Learning keine Berechtigungen konfigurieren. <function-arn>bezieht sich auf eine Ressource für maschinelles Lernen <resource-id>mit der entsprechenden Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.](#)
- [Die Funktion <function-arn>bezieht sich auf Machine Learning Learning-Ressource, für die <resource-id> ResourceAccessPolicy sowohl die Berechtigungen als auch die Ressource fehlen OwnerSetting.](#)
- [Die Funktion <function-arn>bezieht sich auf die Machine Learning Learning-Ressource <resource-id>mit der Berechtigung\ "rw\“, während die Einstellung für den Ressourcenbesitzer GroupPermission nur\ "ro\“ zulässt.](#)
- [NoContainer Die Funktion <function-arn>bezieht sich auf Ressourcen des verschachtelten Zielpfads.](#)
- [Lambda <function-arn> erhält Zugriff auf die Ressource <resource-id>, indem dieselbe Gruppenbesitzer-ID freigegeben wird](#)

Ungültiger MLModel Besitzer — GroupOwnerSetting wird in der ML-Modellressource angegeben, GroupPermission ist aber GroupOwner nicht vorhanden

Lösung: Dieser Fehler wird angezeigt, wenn eine maschinelle Lernressource das [ResourceDownloadOwnerSetting](#) Objekt enthält, die erforderliche GroupPermission Eigenschaft GroupOwner oder aber nicht definiert ist. Um dieses Problem zu beheben, definieren Sie die fehlende Eigenschaft.

NoContainer Die Funktion kann beim Anhängen von Ressourcen für Machine Learning keine Berechtigungen konfigurieren. <function-arn>bezieht sich auf eine Ressource für maschinelles Lernen <resource-id>mit der entsprechenden Berechtigung <ro/rw> in der Ressourcenzugriffsrichtlinie.

Lösung: Sie erhalten diesen Fehler, wenn eine nicht containerisierte Lambda-Funktion Berechtigungen auf Funktionsebene für eine maschinelle Lernressource festlegt. Nicht containerisierte Funktionen müssen Berechtigungen von den Ressourcenbesitzer-Berechtigungen erben, die für die Machine Learning-Ressource definiert sind. Um dieses Problem zu beheben, können Sie die [Berechtigungen des Ressourcenbesitzers \(Konsole\) erben](#) oder [die Berechtigungen aus der Ressourcenzugriffsrichtlinie \(API\) der Lambda-Funktion entfernen](#).

Die Funktion <function-arn>bezieht sich auf Machine Learning Learning-Ressource, für die <resource-id> ResourceAccessPolicy sowohl die Berechtigungen als auch die Ressource fehlen OwnerSetting.

Lösung: Sie erhalten diesen Fehler, wenn die Berechtigungen für die maschinelle Lernressource nicht für die angehängte Lambda-Funktion oder die Ressource konfiguriert sind. Um dieses Problem zu beheben, konfigurieren Sie Berechtigungen in der [ResourceAccessPolicy](#) Eigenschaft für die Lambda-Funktion oder der [OwnerSetting](#) Eigenschaft für die Ressource.

Die Funktion `<function-arn>` bezieht sich auf die Machine Learning Learning-Ressource `<resource-id>` mit der Berechtigung `"rw"`, während die Einstellung für den Ressourcenbesitzer `GroupPermission` nur `"ro"` zulässt.

Lösung: Sie erhalten diesen Fehler, wenn die für die angehängte Lambda-Funktion definierten Zugriffsberechtigungen die für die maschinelle Lernressource definierten Berechtigungen des Ressourcenbesitzers überschreiten. Um dieses Problem zu beheben, legen Sie restriktivere Berechtigungen für die Lambda-Funktion oder weniger restriktive Berechtigungen für den Ressourcenbesitzer fest.

NoContainer Die Funktion `<function-arn>` bezieht sich auf Ressourcen des verschachtelten Zielpfads.

Lösung: Sie erhalten diesen Fehler, wenn mehrere maschinelle Lernressourcen, die an eine nicht containerisierte Lambda-Funktion angehängt sind, denselben Zielpfad oder einen verschachtelten Zielpfad verwenden. Um dieses Problem zu beheben, geben Sie separate Zielpfade für die Ressourcen an.

Lambda `<function-arn>` erhält Zugriff auf die Ressource `<resource-id>`, indem dieselbe Gruppenbesitzer-ID freigegeben wird

Lösung: Sie erhalten diesen Fehler, `runtime.log` wenn dieselbe Betriebssystemgruppe als [Run as Identity der Lambda-Funktion und als Ressourcenbesitzer für eine Machine Learning-Ressource](#) angegeben ist, die Ressource jedoch nicht an die Lambda-Funktion angehängt ist. Diese Konfiguration gewährt der Lambda-Funktion implizite Berechtigungen, mit denen sie ohne AWS IoT Greengrass Autorisierung auf die Ressource zugreifen kann.

Um dieses Problem zu beheben, verwenden Sie eine andere Betriebssystemgruppe für eine der Eigenschaften oder hängen Sie die Ressource für maschinelles Lernen an die Lambda-Funktion an.

AWS IoT Greengrass Kernprobleme bei Docker

Verwenden Sie die folgenden Informationen, um Probleme beim Ausführen eines AWS IoT Greengrass Kerns in einem Docker-Container zu beheben.

Problembereiche

- [Fehler: Unbekannte Optionen: -no-include-email.](#)
- [Warnung: IPv4 ist deaktiviert. Das Netzwerk wird nicht funktionieren.](#)
- [Fehler: Eine Firewall blockiert die Freigabe von Dateien zwischen Fenstern und den Containern.](#)
- [Fehler: Beim Aufrufen der GetAuthorizationToken Operation ist ein Fehler aufgetreten \(AccessDeniedException\): User: arn:aws:iam: :user/ <account-id>ist <user-name>nicht berechtigt,: ecr: on resource: * auszuführen GetAuthorizationToken](#)
- [Fehler: Container kann für den Greengrass-Service nicht erstellt werden: Konflikt. Der Containername „/aws-iot-greengrass“ wird bereits verwendet.](#)
- [Fehler: \[FATAL\] -Fehler beim Zurücksetzen des Mount-Namespaces des Threads aufgrund eines unerwarteten Fehlers: „Vorgang nicht zulässig“. Zur Sicherstellung der Konsistenz wird GGC abstürzen und manuell neu gestartet werden.](#)

Fehler: Unbekannte Optionen: -no-include-email.

Lösung: Dieser Fehler kann auftreten, wenn Sie den Befehl `aws ecr get-login` ausführen. Vergewissern Sie sich, dass Sie die neueste AWS CLI Version installiert haben (zum Beispiel `run: pip install awscli --upgrade --user`). Wenn Sie Windows verwenden und die CLI mit dem MSI-Installationsprogramm installiert haben, müssen Sie den Installationsvorgang wiederholen. Weitere Informationen finden Sie im AWS Command Line Interface Benutzerhandbuch AWS Command Line Interface [unter Installation von unter Microsoft Windows](#).

Warnung: IPv4 ist deaktiviert. Das Netzwerk wird nicht funktionieren.

Lösung: Möglicherweise erhalten Sie diese oder eine ähnliche Meldung, wenn Sie AWS IoT Greengrass auf einem Linux-Computer arbeiten. Aktivieren Sie die IPv4 Netzwerkweiterleitung wie in diesem [Schritt](#) beschrieben. AWS IoT Greengrass Cloud-Bereitstellung und MQTT-Kommunikation funktionieren nicht, wenn die IPv4 Weiterleitung nicht aktiviert ist. Weitere Informationen finden Sie unter [Configure namespaced kernel parameters \(sysctls\) at runtime](#) in der Docker-Dokumentation.

Fehler: Eine Firewall blockiert die Freigabe von Dateien zwischen Fenstern und den Containern.

Lösung: Sie können diese Fehlermeldung oder eine `Firewall Detected`-Nachricht erhalten, wenn Sie Docker auf einem Windows-Computer ausführen. Dies kann auch auftreten, wenn Sie an einem Virtual Private Network (VPN) angemeldet sind und Ihre Netzwerkeinstellungen die Bereitstellung des freigegebenen Laufwerks verhindern. Deaktivieren Sie in diesem Fall das VPN und führen Sie den Docker-Container erneut aus.

Fehler: Beim Aufrufen der `GetAuthorizationToken` Operation ist ein Fehler aufgetreten (`AccessDeniedException`): User: `arn:aws:iam::<account-id>:user/<user-name>` ist nicht berechtigt, `ecr:*` auf `resource:*` auszuführen `GetAuthorizationToken`

Möglicherweise erhalten Sie diesen Fehler, wenn Sie den `aws ecr get-login-password` Befehl ausführen, wenn Sie nicht über ausreichende Berechtigungen für den Zugriff auf ein Amazon ECR-Repository verfügen. Weitere Informationen finden Sie unter [Beispiele für Amazon ECR Repository-Richtlinien](#) und [Zugriff auf ein Amazon ECR-Repository](#) im Amazon ECR-Benutzerhandbuch.

Fehler: Container kann für den Greengrass-Service nicht erstellt werden: Konflikt. Der Containername „/aws-iot-greengrass“ wird bereits verwendet.

Lösung: Dies kann auftreten, wenn der Containername von einem älteren Container verwendet wird. Um dieses Problem zu beheben, führen Sie den folgenden Befehl aus, um den alten Docker-Container zu entfernen:

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

Fehler: [FATAL] -Fehler beim Zurücksetzen des Mount-Namespace des Threads aufgrund eines unerwarteten Fehlers: „Vorgang nicht zulässig“. Zur Sicherstellung der Konsistenz wird GGC abstürzen und manuell neu gestartet werden.

Lösung: Dieser Fehler `runtime.log` kann auftreten, wenn Sie versuchen, eine `GreengrassContainer` Lambda-Funktion für einen AWS IoT Greengrass Kern bereitzustellen, der in einem Docker-Container ausgeführt wird. Derzeit können nur `NoContainer` Lambda-Funktionen in einem Greengrass Docker-Container bereitgestellt werden.

Um dieses Problem zu beheben, [stellen Sie sicher, dass sich alle Lambda-Funktionen im NoContainer Modus befinden](#), und starten Sie eine neue Bereitstellung. Wenn Sie dann den Container starten, binden Sie das vorhandene deployment Verzeichnis nicht an den AWS IoT Greengrass Kern-Docker-Container. Erstellen Sie stattdessen ein leeres deployment-Verzeichnis an seiner Stelle und fügen Sie es über Bind-Mount im Docker-Container ein. Dadurch kann der neue Docker-Container die neueste Bereitstellung mit Lambda-Funktionen empfangen, die im `NoContainer` Modus ausgeführt werden.

Weitere Informationen finden Sie unter [the section called “ AWS IoT Greengrass In einem Docker-Container ausführen”](#).

Fehlerbehebung mit Protokollen

Sie können die Protokollierungseinstellungen für eine Greengrass-Gruppe konfigurieren, z. B. ob Protokolle an Logs gesendet, CloudWatch Protokolle im lokalen Dateisystem gespeichert werden sollen oder beides. Wenn Sie detaillierte Informationen zur Problembehandlung erhalten möchten, können Sie die Protokollierungsstufe vorübergehend zu `DEBUG` ändern. Änderungen an den Protokollierungseinstellungen werden wirksam, wenn Sie die Gruppe bereitstellen. Weitere Informationen finden Sie unter [the section called “Konfigurieren Sie die Protokollierung für AWS IoT Greengrass”](#).

AWS IoT Greengrass Speichert Protokolle auf dem lokalen Dateisystem an den folgenden Orten. Das Lesen der Protokolle auf dem Dateisystem erfordert Root-Berechtigungen.

greengrass-root/`ggc/var/log/crash.log`

Zeigt Meldungen an, die generiert werden, wenn ein AWS IoT Greengrass Kern abstürzt.

greengrass-root/ggc/var/log/system/runtime.log

Zeigt Nachrichten zu den fehlgeschlagenen Komponenten an.

greengrass-root/ggc/var/log/system/

Enthält alle Protokolle von AWS IoT Greengrass Systemkomponenten wie dem Zertifikatsmanager und dem Verbindungsmanager. Mithilfe der Meldungen in ggc/var/log/system/ und sollten Sie herausfinden können ggc/var/log/system/runtime.log, welcher Fehler in AWS IoT Greengrass Systemkomponenten aufgetreten ist.

greengrass-root/ggc/var/log/system/localwatch/

Enthält die Protokolle für die AWS IoT Greengrass Komponente, die das Hochladen von Greengrass-Protokollen in Logs übernimmt. CloudWatch Wenn Sie die Greengrass-Logs nicht einsehen können CloudWatch, können Sie diese Logs zur Fehlerbehebung verwenden.

greengrass-root/ggc/var/log/user/

Enthält alle Protokolle von benutzerdefinierten Lambda-Funktionen. Suchen Sie in diesem Ordner nach Fehlermeldungen von Ihren lokalen Lambda-Funktionen.

Note

Das Standardverzeichnis von /greengrass ist *greengrass-root*. Wenn ein [Schreibverzeichnis](#) konfiguriert wurde, finden Sie auch die Protokolle dort.

Wenn die Protokolle so konfiguriert sind, dass sie in der Cloud gespeichert werden, verwenden Sie CloudWatch Logs, um Protokollmeldungen einzusehen. crash.log ist nur in den Dateisystemprotokollen auf dem AWS IoT Greengrass Kerngerät zu finden.

Wenn AWS IoT es für das Schreiben von Protokollen konfiguriert ist CloudWatch, überprüfen Sie diese Protokolle, falls Verbindungsfehler auftreten, wenn Systemkomponenten versuchen, eine Verbindung herzustellen AWS IoT.

Weitere Hinweise zur AWS IoT Greengrass Protokollierung finden Sie unter [the section called “Überwachen mit AWS IoT Greengrass -Protokollen”](#).

 Note

Protokolle für die AWS IoT Greengrass Core-Software v1.0 werden im *greengrass-root/*`var/log` Verzeichnis gespeichert.

Fehlerbehebung bei Speicherproblemen

Wenn der lokale Dateispeicher voll ist, schlagen einige Komponenten möglicherweise fehl:

- Es werden keine Updates von lokalen Schatten ausgeführt.
- Neue AWS IoT Greengrass Kern-MQTT-Serverzertifikate können nicht lokal heruntergeladen werden.
- Die Bereitstellung schlägt fehl.

Sie sollten immer wissen, wie viel lokaler Speicherplatz verfügbar ist. Sie können den freien Speicherplatz auf der Grundlage der Größe der bereitgestellten Lambda-Funktionen, der Protokollierungskonfiguration (siehe [the section called “Fehlerbehebung mit Protokollen”](#)) und der Anzahl der lokal gespeicherten Schatten berechnen.

Fehlerbehebung für Nachrichten

Alle lokal gesendeten Nachrichten AWS IoT Greengrass werden mit QoS 0 gesendet. AWS IoT Greengrass speichert Nachrichten standardmäßig in einer Warteschlange im Speicher. Daher gehen nicht verarbeitete Nachrichten verloren, wenn der Greengrass-Core neu gestartet wird (z. B. nach einer Gruppenbereitstellung oder einem Gerätereustart). Sie können jedoch AWS IoT Greengrass (v1.6 oder höher) so konfigurieren, dass Nachrichten im Dateisystem zwischengespeichert werden, sodass sie auch nach Kerneustarts bestehen bleiben. Sie können auch die Größe der Warteschlange konfigurieren. Falls Sie eine Warteschlangengröße konfigurieren, stellen Sie sicher, dass sie größer als oder gleich 262144 Byte (256 KB) ist. Andernfalls wird es AWS IoT Greengrass möglicherweise nicht richtig gestartet. Weitere Informationen finden Sie unter [the section called “MQTT-Nachrichtenwarteschlange”](#).

Note

Wenn Sie die standardmäßige speicherinterne Warteschlange verwenden, sollten Sie Gruppen bereitstellen oder das Gerät neu starten, wenn die Serviceunterbrechung möglichst gering ist.

Sie können den Core auch so konfigurieren, dass persistente Sitzungen mit AWS IoT eingerichtet werden. Auf diese Weise kann der Core Nachrichten empfangen, die von dem gesendet wurden, AWS Cloud während der Core offline ist. Weitere Informationen finden Sie unter [the section called “Persistente MQTT-Sitzungen mit AWS IoT Core”](#).

Beheben von Timeout-Problemen während der Schattensynchronisierung

Erhebliche Verzögerungen bei der Kommunikation zwischen einem Greengrass Core-Gerät und der Cloud können dazu führen, dass die Schattensynchronisierung wegen einer Zeitüberschreitung fehlschlägt. In diesem Fall sollten Protokolleinträge angezeigt werden, die etwa wie folgt aussehen:

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
client error: unable to get cloud shadow what_the_thing_is_named for
synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
while awaiting headers)
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
{what_the_thing_is_named VersionDiscontinued []}"
```

Eine mögliche Lösung besteht darin, die Zeitspanne festzulegen, die das Core-Gerät auf eine Antwort des Hosts wartet. Öffnen Sie die Datei [config.json](#) in *greengrass-root*/config, und fügen Sie ein `system.shadowSyncTimeout`-Feld mit einem Zeitüberschreitungswert in Sekunden hinzu. Beispiel:

```
{
  "system": {
```

```
"shadowSyncTimeout": 10
},
"coreThing": {
  "caPath": "root-ca.pem",
  "certPath": "cloud.pem.crt",
  "keyPath": "cloud.pem.key",
  ...
},
...
}
```

Wenn in `config.json` kein `shadowSyncTimeout`-Wert angegeben ist, lautet der Standardwert 5 Sekunden.

 Note

Für die AWS IoT Greengrass Core-Software v1.6 und früher `shadowSyncTimeout` ist die Standardeinstellung 1 Sekunde.

Überprüfen Sie re:POST AWS

Wenn du dein Problem anhand der Informationen zur Fehlerbehebung in diesem Thema nicht lösen kannst, kannst du in re:POST nach verwandten Problemen suchen [Fehlerbehebung](#) oder das [AWS IoT Greengrass Schlagwort auf AWS re:POST](#) nach verwandten Problemen suchen oder eine neue Frage stellen. Mitglieder des AWS IoT Greengrass Teams überwachen AWS re:POST aktiv.

Dokumenthistorie für AWS IoT Greengrass

In der folgenden Tabelle werden wichtige Änderungen am Entwicklerhandbuch nach Juni 2018 beschrieben. AWS IoT Greengrass Um Benachrichtigungen über Aktualisierungen dieser Dokumentation zu erhalten, können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Ende des Supporthinweises für AWS IoT Greengrass Version 1	Hinweis zum Ende des Supports: Am 7. Oktober 2026 AWS wird der Support für eingestellt. AWS IoT Greengrass Version 1 Nach dem 7. Oktober 2026 können Sie nicht mehr auf die Ressourcen zugreifen . AWS IoT Greengrass V1 Weitere Informationen finden Sie unter Migrieren von AWS IoT Greengrass Version 1 .	7. Oktober 2025
Update zum Ende der Unterstützung für v1.11.x Snap	Die Informationen zum Ende des Supports für AWS IoT Greengrass Core v 1.11.x Snap auf snapcraft.io wurden aktualisiert.	22. September 2023
Ende der Unterstützung für v1.11.x Snap	Es wurden Informationen zum Ende des Supports für AWS IoT Greengrass Core v 1.11.x Snap auf snapcraft.io hinzugefügt.	19. September 2023
AWS IoT Greengrass Docker-Images für v1.11.6	Die Docker-Images für die AWS IoT Greengrass Core-Software v1.11.6 sind in Amazon Elastic Container	12. April 2022

Registry (Amazon ECR) und Docker Hub verfügbar. Wir empfehlen, dass Sie immer die neueste Version ausführen.

[AWS IoT Device Tester \(IDT\) für AWS IoT Greengrass V1 veraltete Geräte](#)

IDT für AWS IoT Greengrass V1 generiert keine signierten Qualifikationsberichte mehr.

4. April 2022

[Support-Update für AWS IoT Device Tester für AWS IoT Greengrass](#)

IDT für AWS IoT Greengrass Version 4.4.1 unterstützt jetzt die Verwendung der AWS IoT Greengrass Kernsoftwareversion v1.11.6 für die Gerätequalifizierung.

24. März 2022

[AWS IoT Greengrass Version 1.11.6 veröffentlicht](#)

Version 1.11.6 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden

24. März 2022

[SiteWise IoT-Connector Version 12 veröffentlicht](#)

Version 12 des SiteWise IoT-Connectors ist verfügbar. Diese Version enthält Fehlerkorrekturen.

23. Dezember 2021

[Docker-Images für AWS IoT Greengrass v1.11.5 und v1.10.5](#)

Die Docker-Images für die AWS IoT Greengrass Core-Software v1.11.5 und v1.10.5 sind in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar. Wir empfehlen, dass Sie immer die neueste Version ausführen.

22. Dezember 2021

[AWS IoT Greengrass V1 Wartungsrichtlinie](#)

Die AWS IoT Greengrass V1 Wartungsrichtlinie definiert die verschiedenen Wartungs- und Aktualisierungsstufen für den AWS IoT Greengrass V1 Service und die AWS IoT Greengrass Kernsoftware v1.x.

20. Dezember 2021

[AWS IoT Device Tester Version 4.4.1 veröffentlicht](#)

IDT für AWS IoT Greengrass Version 4.4.1 ist jetzt verfügbar. Diese Version enthält die AWS IoT Greengrass Qualification Suite (GGQ) v1.3.1 und unterstützt die Verwendung der AWS IoT Greengrass Kernsoftwareversionen v1.11.5 und v1.10.5 für die Gerätequalifizierung.

20. Dezember 2021

[AWS IoT Greengrass
Versionen 1.11.5 und 1.10.5
veröffentlicht](#)

Die Versionen 1.11.5 und 1.10.5 der Core-Software sind verfügbar. AWS IoT Greengrass Diese Versionen enthalten Leistungsverbesserungen und Fehlerkorrekturen. Wir empfehlen, dass Sie immer die neueste Version ausführen.

12. Dezember 2021

[Die Docker-Images AWS
IoT Greengrass v1.11.4
und v1.10.4 wurden erneut
veröffentlicht](#)

Docker-Images für die AWS IoT Greengrass Core-Softwareversionen 1.11.4 und 1.10.4 wurden erneut auf Amazon Elastic Container Registry (Amazon ECR) und Docker Hub veröffentlicht, um Bugfixes mit zu beheben. BusyBox Verwenden Sie die Tags oder, um die neuesten Docker-Images zu verwenden. 1.11.4-1 1.10.4-1 Weitere Informationen zu verfügbaren Tags finden Sie unter [amazon/aws-iot-greengrass](#) in Docker Hub.

8. Dezember 2021

[CloudWatch Der Metrics
Connector unterstützt doppelte
Zeitstempel in Eingabedaten](#)

Sie können jetzt Eingabedaten mit doppelten Zeitstempeln an diesen Konnektor senden.

19. November 2021

[Update zur Vermeidung des Problems des verwirrten Stellvertreters \(dienstübergreifend\)](#)

AWS IoT Greengrass unterstützt die Verwendung der Kontextschlüssel [aws:SourceArn](#) und der [aws:SourceAccount](#) globalen Bedingungsschlüssel in IAM-Ressourcenrichtlinien, um das Problem des verwirrten Stellvertreters zu vermeiden.

1. November 2021

[Docker-Images für v1.11.4 AWS IoT Greengrass](#)

Die Docker-Images für die AWS IoT Greengrass Core-Software v1.11.4 sind in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar. Wir empfehlen, dass Sie immer die neueste Version ausführen.

24. August 2021

[Hat AWS IoT Greengrass v1.11.4 Snap veröffentlicht](#)

Version 1.11.4 des Snaps ist verfügbar. AWS IoT Greengrass Wir empfehlen, dass Sie immer die neueste Version ausführen.

20. August 2021

[Support-Update für AWS IoT Device Tester für AWS IoT Greengrass](#)

IDT für AWS IoT Greengrass Version 4.1.0 unterstützt jetzt die Verwendung der AWS IoT Greengrass Kernsoftwareversion v1.11.4 für die Gerätequalifizierung.

18. August 2021

[AWS IoT Greengrass Version 1.11.4 veröffentlicht](#)

Version 1.11.4 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version behebt ein Problem mit dem Stream Manager, das Upgrades von einer früheren Version der Core-Software auf v1.11.3 verhinderte. AWS IoT Greengrass Wir empfehlen, dass Sie immer die neueste Version ausführen.

17. August 2021

[VPC-Endpunkte \(AWS PrivateLink\)](#)

AWS IoT Greengrass unterstützt jetzt Schnittstellen-VPC-Endpunkte (AWS PrivateLink) für die AWS IoT Greengrass Steuerungsebene. Sie können eine private Verbindung zwischen Ihrer VPC und der AWS IoT Greengrass Kontrollebene herstellen.

16. August 2021

[AWS IoT Device Tester Version 4.1.0 veröffentlicht](#)

Version 4.1.0 von AWS IoT Device Tester für AWS IoT Greengrass ist verfügbar. Diese Version unterstützt die Verwendung der AWS IoT Greengrass Kernsoftwareversionen 1.11.3 und 1.10.4 für die Gerätequalifizierung.

23. Juni 2021

[Snap v1.11.3 veröffentlicht](#)
[AWS IoT Greengrass](#)

Version 1.11.3 des AWS IoT Greengrass Snap enthält Leistungsverbesserungen und Fehlerkorrekturen. Als bewährte Methode empfehlen wir, immer die neueste Version zu verwenden.

15. Juni 2021

[Docker-Images für AWS](#)
[IoT Greengrass v1.11.3 und](#)
[v1.10.4 veröffentlicht](#)

Die Docker-Images für die AWS IoT Greengrass Core-Software v1.11.3 und v1.10.4 sind in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar. Diese Versionen von AWS IoT Greengrass Core enthalten Leistungsverbesserungen und Fehlerkorrekturen. Wir empfehlen, dass Sie immer die neueste Version ausführen.

15. Juni 2021

[AWS IoT Greengrass Version](#)
[1.11.3 veröffentlicht](#)

Version 1.11.3 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden

14. Juni 2021

<u>AWS IoT Greengrass Version 1.10.4 veröffentlicht</u>	Version 1.10.4 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden.	14. Juni 2021
<u>Version 2 des Modbus-TCP-Protokolladapters veröffentlicht</u>	Version 2 des Modbus-TCP-Protokolladapter-Connectors ist verfügbar. Diese Version fügte Unterstützung für ASCII- und ISO8859 kodierte UTF8 Quellzeichenfolgen hinzu.	24. Mai 2021
<u>Version 7 des Connectors für die Bereitstellung von Docker-Anwendungen veröffentlicht</u>	Version 7 des Greengrass Docker Application Deployment Connectors ist verfügbar.	05. April 2021
<u>AWS IoT Greengrass Version 1.11.1 veröffentlicht</u>	Version 1.11.1 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden.	29. März 2021

[AWS IoT Device Tester
Version 4.0.2 wurde veröffent
licht](#)

Version 4.0.2 von AWS IoT Device Tester für AWS IoT Greengrass ist verfügbar. Diese Version ersetzt IDT v4.0.0 und bietet Unterstützung für Version 1.11.1 der Core-Software. AWS IoT Greengrass Dies behebt auch ein Problem, das dazu führte, dass IDT Fehler bei der Hardwaresicherheitsintegration (HSI) maskierte.

29. März 2021

[SiteWise IoT-Connector
Version 11 veröffentlicht](#)

Version 11 des SiteWise IoT-Connectors ist verfügbar. Dadurch wird die Unterstützung für Zeichenketten gestartet, die versteckte oder nicht druckbare Zeichen enthalten. Diese Version enthält auch allgemeine Leistungsverbesserungen und Fehlerkorrekturen.

24. März 2021

[Der AWS IoT Greengrass
Snap-Version 1.11.0 wurde
erneut veröffentlicht](#)

AWS IoT Greengrass Snap-Version 1.11.0 wurde auf Snapcraft erneut veröffentlicht, um Bugfixes und einen möglichen Anwendungssabsturz bei der Verwendung des Python-Interpreters zu beheben. AWS IoT Greengrass bietet keine Snaps für die Softwareversionen 1.10 und 1.9.

19. März 2021

[Support-Update für AWS IoT Device Tester für AWS IoT Greengrass](#)

IDT für AWS IoT Greengrass Version 4.0.0 unterstützt jetzt die Verwendung der AWS IoT Greengrass Kernsoftwareversion v1.10.3 für die Gerätequalifizierung.

18. März 2021

[Der AWS IoT Greengrass Snap-Version v1.8.4 wurde erneut veröffentlicht](#)

AWS IoT Greengrass Snap-Version 1.8.4 wurde auf Snapcraft erneut veröffentlicht, um Bugfixes und einen möglichen Anwendungsabsturz bei der Verwendung des Python-Interpreters zu beheben.

15. März 2021

[Das Docker-Image v1.11.0 für ARMv7L wurde erneut veröffentlicht AWS IoT Greengrass](#)

Das Docker-Image für die AWS IoT Greengrass Core-Softwareversion 1.11.0 für die ARMv7L-Plattform wurde in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub erneut veröffentlicht, um Bugfixes und einen möglichen Anwendungsabsturz bei der Verwendung des Python-Interpreters zu beheben.

8. März 2021

[AWS IoT Greengrass Docker-Images v1.10.3 veröffentlicht](#)

Docker-Images für die AWS IoT Greengrass Core-Softwareversion 1.10.3 sind jetzt in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub verfügbar.

8. März 2021

[Die Docker-Images v1.11.0 und v1.9.4 wurden erneut veröffentlicht AWS IoT Greengrass](#)

Docker-Images für die AWS IoT Greengrass Core-Softwareversionen 1.11.0 und 1.9.4 wurden in Amazon Elastic Container Registry (Amazon ECR) und Docker Hub erneut veröffentlicht, um Bugfixes und einen möglichen Anwendungsabsturz bei der Verwendung des Python-Interpreters zu beheben. Die Docker-Images für ARMv7l wurden derzeit nicht erneut veröffentlicht.

26. Februar 2021

[AWS IoT Greengrass Version 1.10.3 veröffentlicht](#)

Version 1.10.3 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version fügt die `systemComponentAuthTimeout`-Core-Konfigurationseigenschaft hinzu und enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden.

24. Februar 2021

[SiteWise IoT-Connector Version 10 veröffentlicht](#)

Version 10 des SiteWise IoT-Connectors ist verfügbar. Diese Version behebt Stabilitätsprobleme mit dem StreamManager Client, wenn die Verbindung unterbrochen wird, und verbessert die Verarbeitung von OPC-UA-Werten, wenn a SourceTimestamp fehlt. Verwenden Sie den SiteWise IoT-Connector, um lokale Geräte- und Gerätedaten an Anlageneigenschaften im IoT zu senden SiteWise.

22. Januar 2021

[SiteWise IoT-Connector Version 9 veröffentlicht](#)

Version 9 des SiteWise IoT-Connectors ist verfügbar. Damit werden benutzerdefinierte StreamManager Greengrass-Stream-Ziele, OPC-UA-Deadbanding, benutzerdefinierter Scanmodus und benutzerdefinierte Scanrate unterstützt. Dazu gehört auch eine verbesserte Leistung bei Konfigurationsupdates, die vom SiteWise IoT-Gateway aus vorgenommen werden. Verwenden Sie den SiteWise IoT-Connector, um lokale Geräte- und Gerätedaten an Anlageneigenschaften im IoT zu senden SiteWise.

15. Dezember 2020

[AWS IoT Device Tester
Version 4.0.0 veröffentlicht](#)

Version 4.0.0 von AWS IoT Device Tester für AWS IoT Greengrass ist verfügbar. Mit dieser Version können Sie IDT verwenden, um Ihre benutzerdefinierten Testsuiten für die Gerätevalidierung zu entwickeln und auszuführen. Dazu gehören auch codesignierte IDT-Anwendungen für macOS und Windows.

15. Dezember 2020

[AWS IoT Greengrass Snap
v1.11](#)

Version 1.11.0 des AWS IoT Greengrass Snap unterstützt nicht containerisierte Lambda-Funktionen. Als bewährte Methode empfehlen wir, immer die neueste Version zu verwenden.

6. Dezember 2020

[SiteWise IoT-Connector
Version 8 veröffentlicht](#)

Version 8 des SiteWise IoT-Connectors ist verfügbar. Diese Version verbessert die Stabilität, wenn die Netzwerkonnektivität des Connectors unterbrochen wird. Verwenden Sie den SiteWise IoT-Connector, um lokale Geräte- und Gerätedaten an Anlageneigenschaften im IoT zu senden SiteWise.

19. November 2020

[Der Kinesis Firehose-
Anschluss unterstützt den
Modus „Kein Container“](#)

Sie können den Isolation Mode Parameter verwenden , um den Containerisierungsmodus für den Konnektor zu konfigurieren.

19. Oktober 2020

[Connector für die Bereitstellung von Docker-Anwendungen, Version 6 veröffentlicht](#)

Version 6 des Greengrass Docker Application Deployment Connectors ist verfügbar.

18. September 2020

[AWS IoT Greengrass Version 1.11.0 veröffentlicht](#)

Version 1.11.0 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version fügt die Telemetriefunktion zur Systemintegrität und eine lokale Integritätsprüfungs-API hinzu. Stream Manager kann jetzt Daten nach Amazon Simple Storage Service (Amazon S3) und IoT exportieren SiteWise. Diese Version enthält auch Leistungsverbesserungen und Fehlerkorrekturen. Wir empfehlen, dass Sie immer die neueste Version ausführen.

16. September 2020

[SiteWise IoT-Connector Version 7 veröffentlicht](#)

Version 7 des SiteWise IoT-Connectors ist verfügbar. Diese Version behebt ein Problem mit Gateway-Metriken. Verwenden Sie den SiteWise IoT-Connector, um lokale Geräte- und Gerätedaten an Anlageneigenschaften im IoT zu senden SiteWise.

14. August 2020

[ServiceNow MetricBase Die Konnektoren Integration, Splunk Integration und Twilio Notifications unterstützen den Modus „Kein Container“](#)

Sie können den Isolation Mode Parameter verwenden , um den Containerisierungsmodus für den Connector zu konfigurieren.

30. Juli 2020

[Der SNS-Connector unterstützt den Modus „Kein Container“](#)

Sie können den `Isolation Mode` Parameter verwenden , um den Containerisierungsmodus für den Konnektor zu konfigurieren.

6. Juli 2020

[CloudWatch Der Metrics-Connector unterstützt den Modus „Kein Container“](#)

Sie können den `Isolation Mode` Parameter verwenden , um den Containerisierungsmodus für den Konnektor zu konfigurieren.

17. Juni 2020

[AWS IoT Greengrass Version 1.10.2 veröffentlicht](#)

Version 1.10.2 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version fügt die `mqttOperationTimeout` -Core-Eigenschaft hinzu und enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden .

08. Juni 2020

[Die Tensorflow-Installationsprogramme für maschinelles Lernen sind veraltet](#)

AWS IoT Greengrass Die vorkonfigurierten Tensorflow-Installationsprogramme für maschinelles Lernen sind veraltet. Machine-Learning-Beispiele wurden auf Python 3.7 aktualisiert.

29. Mai 2020

[Chainer-Framework-Unterstützung und Greengrass-Installationsprogramme für maschinelles Lernen sind veraltet](#)

AWS IoT Greengrass vorkonfigurierte Installationsprogramme und Downloads für maschinelles Lernen für und DLR sind veraltet. MXNet Unterstützung für Chainer-Framework und zugehörige Downloads ist veraltet.

4. Mai 2020

[SiteWise IoT-Connector Version 6 veröffentlicht](#)

Version 6 des SiteWise IoT-Connectors ist verfügbar. Diese Version bietet Unterstützung für CloudWatch Metriken und die automatische Erkennung neuer OPC-UA-Tags. Dies bedeutet, dass Sie Ihr Gateway nicht neu starten müssen, wenn sich Tags für Ihre OPC-UA-Quellen ändern. Diese Version des Connectors erfordert Stream Manager und AWS IoT Greengrass Core-Software v1.10.0 oder höher. Verwenden Sie den SiteWise IoT-Connector, um lokale Geräte- und Gerätedaten an Anlageneigenschaften im IoT zu senden SiteWise.

29. April 2020

[Konnektoren wurden auf Python 3.7 aktualisiert](#)

Konnektoren, die die Python-Laufzeit unterstützen, wurden auf Python 3.7 aktualisiert. Wir empfehlen, Ihre Konnektor-Versionen von Python 2.7 auf Python 3.7 zu aktualisieren.

29. April 2020

<u>Das Greengrass-Geräte-Setup kann im unbeaufsichtigten Modus ausgeführt werden</u>	Sie können Greengrass Device Setup im Hintergrundmodus ausführen, damit das Skript Sie nicht zur Eingabe von Werten auffordert.	27. April 2020
<u>Neue Docker-Basis-Images</u>	Sie können AWS IoT Greengrass Docker-Images herunterladen, die auf Alpine Linux-Basisimages (x86_64, ARMv7L oder) basieren. AArch64	23. April 2020
<u>AWS IoT Greengrass Version 1.10.1 veröffentlicht</u>	Version 1.10.1 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wir empfehlen, immer die neueste Version zu verwenden.	16. April 2020
<u>Neues Kapitel bezüglich der Sicherheit</u>	AWS IoT Greengrass Der Sicherheitsinhalt wurde neu organisiert und es wurden neue Informationen hinzugefügt.	30. März 2020
<u>Verwenden Sie den APT-Paketmanager zur Installation AWS IoT Greengrass</u>	Auf unterstützten Debian-basierten Linux-Distributionen können Sie die AWS IoT Greengrass Core-Software apt zur Installation auf Ihren Geräten verwenden.	26. Februar 2020

[SiteWise IoT-Connector Version 5 veröffentlicht](#)

Version 5 des SiteWise IoT-Connectors ist verfügbar. Diese Version behebt ein Kompatibilitätsproblem mit der AWS IoT Greengrass Core-Software v1.9.4. Verwenden Sie den SiteWise IoT-Connector, um lokale Geräte- und Gerätedaten an Anlageneigenschaften im IoT zu senden SiteWise.

12. Februar 2020

[Neues Skript zum schnellen Einrichten eines Kerngeräts](#)

Mit Greengrass Device Setup lässt sich Ihr Core-Gerät in wenigen Minuten konfigurieren. Unterstützt AWS IoT Greengrass jetzt auch die Lambda-Funktionen von Node.js 12.x.

20. Dezember 2019

[AWS IoT Greengrass Version 1.10.0 veröffentlicht](#)

Version 1.10.0 der AWS IoT Greengrass Core-Software ist verfügbar. Zu den neuen Funktionen in dieser Version gehören Stream Manager, Containerunterstützung mit dem Docker Application Deployment Connector, nicht containerisierte Lambda-Funktionen, die auf Ressourcen für maschinelles Lernen zugreifen können, Unterstützung für persistente MQTT-Sitzungen mit AWS IoT und Unterstützung für lokalen MQTT-Verkehr über einen bestimmten Port.

25. November 2019

[Konsolenunterstützung für Bereitstellungsbenachrichtigungen](#)

Verwenden Sie die EventBridge Amazon-Konsole, um Ereignisregeln zu erstellen, die ausgelöst werden, wenn sich der Status Ihrer Greengrass-Gruppenbereitstellungen ändert.

14. November 2019

[AWS IoT Greengrass Version 1.9.4 veröffentlicht](#)

Version 1.9.4 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Als bewährte Methode empfehlen wir, dass Sie immer die neueste Version ausführen.

17. Oktober 2019

[Konsolenunterstützung für die Verwaltung der Greengrass-Servicerolle](#)

Verwenden Sie neue und verbesserte Funktionen in der AWS IoT Konsole, um Ihre Greengrass-Servicerolle zu verwalten.

4. Oktober 2019

[Konsolenunterstützung für die Verwaltung von Tags auf Gruppenebene](#)

In der AWS IoT Konsole können Sie Tags für Ihre Greengrass-Gruppen erstellen , anzeigen und verwalten.

23. September 2019

[Neue Konnektoren für maschinelles Lernen](#)

Verwenden Sie den ML-Feedback-Konnektor, um Modelleingaben und -voraussagen zu veröffentlichen, und den ML-Objekterkennungs-Konnektor, um einen lokalen Objekterkennungs-Inferenzservice auszuführen.

19. September 2019

[AWS IoT Greengrass Version 1.9.3 veröffentlicht](#)

Version 1.9.3 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version ermöglicht die Installation der AWS IoT Greengrass Core-Software auf Raspbian-Distributionen auf ARMv6L-Architekturen, unterstützt OTA-Updates auf Port 443 mit ALPN und enthält einen Bugfix für binäre Payloads, die von Python 2.7 Lambda-Funktionen an andere Lambda-Funktionen gesendet wurden.

12. September 2019

[AWS IoT Greengrass Version
1.8.4 veröffentlicht](#)

Version 1.8.4 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält Leistungsverbesserungen und Fehlerbehebungen. Wenn Sie v1.8.x ausführen, sollten Sie ein Upgrade auf v1.8.4 oder v1.9.3 ausführen. Im Fall älterer Versionen sollten Sie ein Upgrade auf v1.9.3 ausführen.

30. August 2019

[AWS IoT Greengrass Version
1.9.2 veröffentlicht mit
Unterstützung für OpenWrt](#)

Version 1.9.2 der AWS IoT Greengrass Core-Software ist verfügbar. Mit dieser Version können Sie die AWS IoT Greengrass Core-Software auf OpenWrt Distributionen mit Armv8 (AArch64) - und ARMv7L-Architekturen installieren.

20. Juni 2019

[AWS IoT Greengrass Version
1.8.3 veröffentlicht](#)

Version 1.8.3 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält allgemeine Leistungsverbesserungen und Fehlerbehebungen. Wenn Sie v1.8.x ausführen, empfehlen wir, ein Upgrade auf v1.8.3 oder v1.9.2 durchzuführen. Für ältere Versionen empfehlen wir, ein Upgrade auf v1.9.2 durchzuführen.

20. Juni 2019

[AWS IoT Greengrass Version
1.9.1 veröffentlicht](#)

Version 1.9.1 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält einen Bugfix für Nachrichten von AWS IoT , die im Thema ein Platzhalt erzeichen enthalten.

10. Mai 2019

[AWS IoT Greengrass Version
1.8.2 veröffentlicht](#)

Version 1.8.2 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält allgemeine Leistungsverbesserungen und Fehlerbehebungen. Wenn Sie v1.8.x ausführen, empfehlen wir, dass Sie ein Upgrade auf v1.8.2 oder v1.9.0 durchführen. Für ältere Versionen empfehlen wir, dass Sie ein Upgrade auf v1.9.0 durchführen.

2. Mai 2019

[AWS IoT Greengrass Version
1.9.0 veröffentlicht](#)

Neue Funktionen: Support für Lambda-Laufzeiten von Python 3.7 und Node.js 8.10, optimierte MQTT-Verbindungen und Elliptic Curve (EC) -Schlüsselunterstützung für den lokalen MQTT-Server.

1. Mai 2019

[AWS IoT Greengrass Version 1.8.1 veröffentlicht](#)

Version 1.8.1 der AWS IoT Greengrass Core-Software ist verfügbar. Diese Version enthält allgemeine Leistungsverbesserungen und Fehlerbehebungen. Als bewährte Methode empfehlen wir, dass Sie immer die neueste Version ausführen.

18. April 2019

[AWS IoT Greengrass Snap ist auf Snapcraft verfügbar](#)

Verwenden Sie die AWS IoT Greengrass Snap Store-App, um schnell Software auf Linux-Geräten mit AWS IoT Greengrass zu entwerfen, zu testen und bereitzustellen.

1. April 2019

[Support für mehr Zugriffskontrolle mithilfe von Tag-basierten Berechtigungen](#)

Sie können Tags in AWS Identity and Access Management (IAM-) Richtlinien verwenden, um den Zugriff auf Ihre AWS IoT Greengrass Ressourcen zu kontrollieren.

29. März 2019

[IoT Analytics Analytics-Konnektor veröffentlicht](#)

Verwenden Sie den IoT Analytics Analytics-Konnektor, um lokale Gerätedaten an AWS IoT Analytics Kanäle zu senden.

15. März 2019

[Batch-Unterstützung im Kinesis Firehose-Anschluss](#)

Der Kinesis Firehose-Connector unterstützt das Senden von Batch-Datensätzen an Amazon Data Firehose in einem bestimmten Intervall.

15. März 2019

[CloudFormationAWS IoT Greengrass Unterstützung für Ressourcen](#)

Verwenden Sie CloudFormation Vorlagen, um AWS IoT Greengrass Ressourcen zu erstellen und zu verwalten.

15. März 2019

[AWS IoT Greengrass Version 1.8.0 veröffentlicht](#)

Neue Funktionen: Konfigurierbare Standardzugriffsid-entität für Lambda-Funktionen, Unterstützung für HTTPS-Verkehr über Port 443 und vorhersehbar benannter Client IDs für MQTT-Verbindungen mit AWS IoT

7. März 2019

[AWS IoT Greengrass Versionen 1.7.1 und 1.6.1 veröffentlicht](#)

Die Versionen 1.7.1 und 1.6.1 der AWS IoT Greengrass Core-Software sind verfügbar. Diese Versionen erfordern Linux-Kernel-Version 3.17 oder höher. Wir empfehlen Kunden, die eine Version der Greengrass Core-Software nutzen, sofort ein Upgrade auf Version 1.7.1 auszuführen.

11. Februar 2019

[SageMaker AI Neo Deep-Learning-Laufzeit](#)

Die SageMaker AI Neo Deep Learning-Laufzeit unterstützt Modelle für maschinelles Lernen, die vom SageMaker AI Neo Deep Learning-Compiler optimiert wurden.

28. November 2018

[AWS IoT Greengrass In einem Docker-Container ausführen](#)

Sie können AWS IoT Greengrass in einem Docker-Container ausführen, indem Sie Ihre Greengrass-Gruppe so konfigurieren, dass sie ohne Containerisierung ausgeführt wird.

26. November 2018

[AWS IoT Greengrass Version 1.7.0 veröffentlicht](#)

Neue Funktionen: Greengrass-Konnektoren, Local Secrets Manager, Isolations- und Berechtigungseinstellungen für Lambda-Funktionen, Hardware-Root-of-Trust-Sicherheit, Verbindung über ALPN oder Netzwerk-Proxy und Raspbian Stretch-Unterstützung.

26. November 2018

[AWS IoT Greengrass Software-Downloads](#)

Die AWS IoT Greengrass Core-Software-, AWS IoT Greengrass Core SDK- und AWS IoT Greengrass Machine Learning Learning-SDK-Pakete können über Amazon CloudFront heruntergeladen werden.

26. November 2018

[AWS IoT Gerätetester für AWS IoT Greengrass](#)

Verwenden Sie AWS IoT Device Tester, AWS IoT Greengrass um zu überprüfen, ob Ihre CPU-Architektur, Kernel-Konfiguration und Treiber damit funktionieren AWS IoT Greengrass.

26. November 2018

[AWS CloudTrail Protokollierung von AWS IoT Greengrass API-Aufrufen](#)

AWS IoT Greengrass ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS IoT Greengrass.

29. Oktober 2018

[Support für TensorFlow v1.10.1 auf NVIDIA Jetson TX2](#)

Die TensorFlow bereitgestellte vorkompilierte Bibliothek für NVIDIA Jetson verwendet jetzt TX2 v1.10.1 AWS IoT Greengrass . TensorFlow Dadurch werden Jetpack 3.3 und CUDA Toolkit 9.0 unterstützt.

18. Oktober 2018

[Support für MXNet v1.2.1-Ressourcen für maschinelles Lernen](#)

AWS IoT Greengrass unterstützt Modelle für maschinelles Lernen, die mit MXNet Version 1.2.1 trainiert wurden.

29. August 2018

[AWS IoT Greengrass Version 1.6.0 veröffentlicht](#)

Neue Funktionen: Ausführbare Lambda-Dateien, konfigurierbare Nachrichtenwarteschlange, konfigurierbares Intervall für erneute Verbindungsversuche, Volume-Ressourcen unter /proc und konfigurierbares Schreibverzeichnis.

26. Juli 2018

Frühere Aktualisierungen

In der folgenden Tabelle werden wichtige Änderungen am Entwicklerhandbuch vor Juli 2018 beschrieben. AWS IoT Greengrass

Änderungen	Beschreibung	Date
AWS IoT Greengrass Version 1.5.0 veröffentlicht	Neue Funktionen: • Lokale Machine Learning-Inferenz mit Cloud-geschulten Modellen durchführen. Weitere Informationen finden Sie unter Durchführen von Machine Learning-Inferenzen. • Die Lambda-Funktionen von Greengrass unterstützen neben JSON auch binäre Eingabedaten. Weitere Informationen finden Sie unter AWS IoT Greengrass Core-Versionen.	29. März 2018
AWS IoT Greengrass Version 1.3.0 veröffentlicht	Neue Funktionen: • Over-the-air (OTA) Update-Agent, der in der Cloud bereitgestellte Greengrass-Aktualisierungsaufträge verarbeiten kann. Weitere Informationen finden Sie unter OTA-Updates der AWS IoT Greengrass Core-Software. • Greifen Sie über die Funktionen von Greengrass Lambda auf lokale Peripheriegeräte und Ressourcen zu. Weitere Informationen finden Sie unter Greifen Sie mit Lambda-Funktionen und Konnektoren auf lokale Ressourcen zu.	27. November 2017
AWS IoT Greengrass Version 1.1.0 veröffentlicht	Neue Funktionen: • Setzen Sie die bereitgestellten AWS IoT Greengrass Gruppen zurück. Weitere Informationen finden Sie unter Zurücksetzen von Bereitstellungen. • Support für Node.js 6.10- und Java 8 Lambda-Laufzeiten zusätzlich zu Python 2.7.	20. September 2017

Änderungen	Beschreibung	Date
AWS IoT Greengrass Version 1.0.0 veröffentlicht	AWS IoT Greengrass ist allgemein verfügbar.	7. Juni 2017